ServletS SyllabuS (5 uNItS)

OCWCD Bits

Detailed Information

Unit-1: The Servlet Technology Model

Objectives

- For each of the HTTP Methods (such as GET, POST, HEAD, and so on) describe the purpose of the method and the technical characteristics of the HTTP Method protocol, list triggers that might cause a Client (usually a Web browser) to use the method; and identify the HttpServlet method that corresponds to the HTTP Method.
- Using the HttpServletRequest interface, write code to retrieve HTML form parameters from the request, retrieve HTTP request header information, or retrieve cookies from the request.
- Using the HttpServletResponse interface, write code to set an HTTP response header, set the content type of the response, acquire a text stream for the response, acquire a binary stream for the response, redirect an HTTP request to another URL, or add cookies to the response.
- Describe the purpose and event sequence of the servlet life cycle: (1) servlet class loading, (2) servlet instantiation, (3) call the init method, (4) call the service method, and (5) call destroy method.

Unit- 2: The Structure and Deployment of Web Applications

Objectives

- Construct the file and directory structure of a Web Application that may contain
  - static content,
  - JSP pages,
  - servlet classes,
  - the deployment descriptor,
  - tag libraries,
  - JAR files, and
  - Java class files; and describe how to protect resource files from HTTP access.
- Describe the purpose and semantics of the deployment descriptor.
- Construct the correct structure of the deployment descriptor.
- Explain the purpose of a WAR file and describe the contents of a WAR file, how one may be constructed

Unit-4: Session Management

Objectives

- Write servlet code to store objects into a session object and retrieve objects from a session object.
- Given a scenario describe the APIs used to access the session object, explain when the session object was created, and describe the mechanisms used to destroy the session object, and when it was destroyed.
- Using session listeners, write code to respond to an event when an object is added to a session, and write code to respond to an event when a session object migrates from one VM to another.
- Given a scenario, describe which session management mechanism the Web container could employ, how cookies might be used to manage sessions, how URL rewriting might be used to manage sessions, and write servlet code to perform URL rewriting.

Unit-5 : Web Application Security

Objectives

- Based on the servlet specification, compare and contrast the following security mechanisms:
  - Authentication,
  - Authorization,
  - Data Integrity, and
  - Confidentiality

- In the deployment descriptor, declare a security constraint, a Web resource, the transport guarantee, the login configuration, and a security role.

- Compare and contrast the authentication types (BASIC, DIGEST, FORM, and CLIENT-CERT); describe how the type works; and given a scenario, select an appropriate type.

Advanced Java

With Core Java knowledge we can develop Stand Alone Applications.

The applications which are running on a single machine are called stand alone applications.
Eg: Calculator, MS Word
Any Core Java application
If we want to develop web applications then we should go for Advanced Java.

The applications which are providing services over the web, are called web applications

Eg: durgasoftvideos.com,gmail.com,facebook.com,durgasoft.com

In Java we can develop web applications by using the following technologies...

JDBC
Servlets
JSPs

Where ever Presentation logic is required i.e to display something to the end user then we should go for JSP. i.e JSP meant for View Component.

Eg: display login page
display inbox page
display error page
display result page
etc..

Where ever some processing logic is required then we should go for Servlet. i.e Servlet meant for Processing Logic/Business Logic. Servlet will always work internally.

Eg: Verify User
Communicate with database
Process end user's data
etc..

From Java application(Normal Java class or Servlet) if we want to communicates with Database then we should go for JDBC.

Eg: To get Astrology information from database
To get mails information from database

In Java there are 3 editions are available

1. Java Standard Edition (JSE | J2SE)
2. Java Enterprise Edition(JEE | J2EE)
3. Java Micro Edition (JME | J2ME)

JDBC is the part of JSE

Servlets and JSPs are the part of JEE

Current version of JSE is Java 1.8
Current version of JDBC is :4.2V

Current version of JEE is 7.0
Current version of Servlets: 3.1V
Current version of JSP is: 2.3V

Current versions are:
JDBC 4.2V
Servlets 3.1V
JSPs 2.3V

Unit 1: The Servlet Technology Model

Web application:

The application which is developed only by using web related technologies(like html,xml,java script,angular js,servlets,jsps etc)is called web application.

eg: durgasoftvideos.com,gmail.com,durgasoft.com,facebook.com etc..

Web application can provide services over the web to the end users.

Web server:

Once we developed web application ,we required special software to run that application.This special software is nothing but web server.

Web server provides environment to run web applications.

Eg:
Tomcat
Oracle Http Server(OHS)
Resin
JETTY
Lighttpd..

Deployment and Undeployment:

The process of placing web application inside web server is called deployment.

The process of removing web application from the web server is called undeployment.

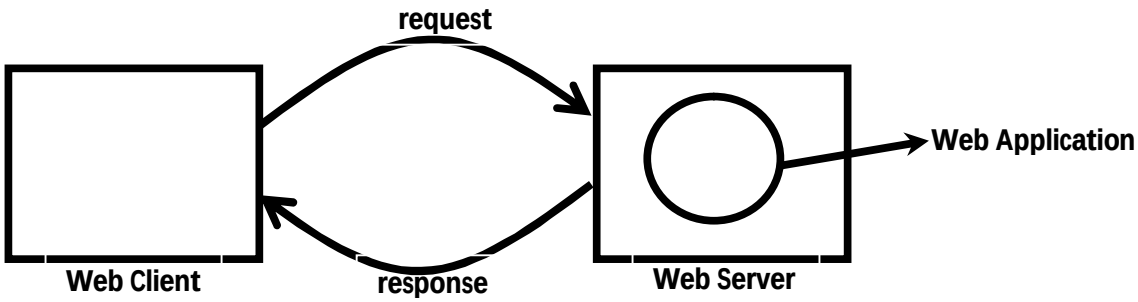
Web client:

It is also known as User-Agent.To send a request to web server a special software is required which is nothing but web client.i.e By using web client we can communicate with web server.

Eg:
The most commonly used web client is browser,which is GUI Based(chrome,mozilla..)

There are several CUI Based web clients also

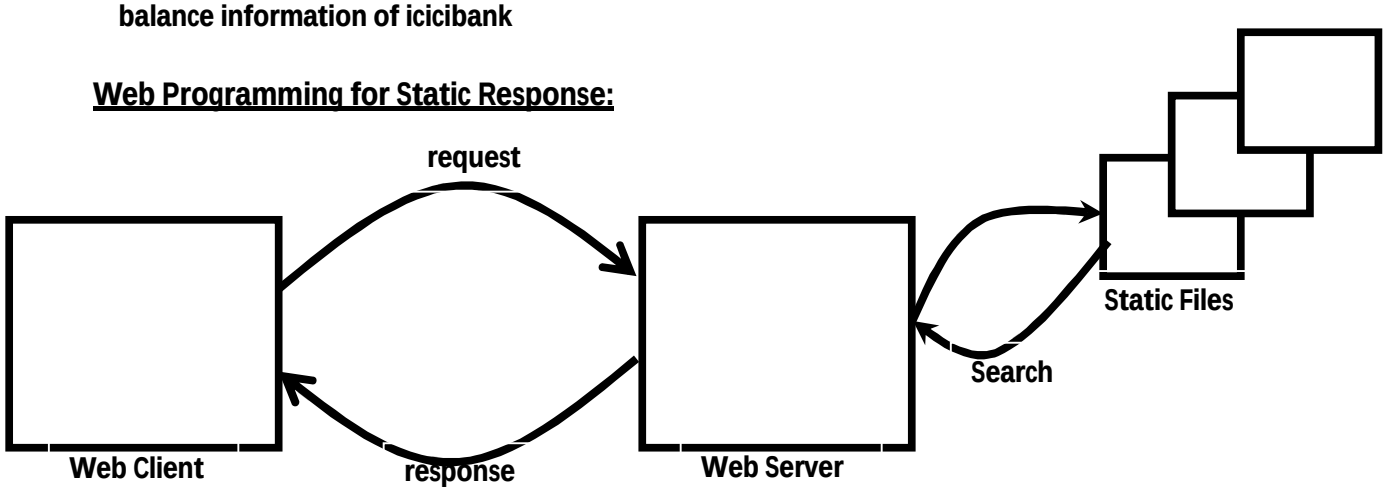
CURL,lynx etc



Static Response vs Dynamic Response:

The response which won't be changed from person to person and time to time,such type of response is called static response.
eg: login page of gmail
home page of icicibank
The response which is varied from person to person and time to time,such type of response is called dynamic response.
eg: inbox page of gmail
balance information of icicibank

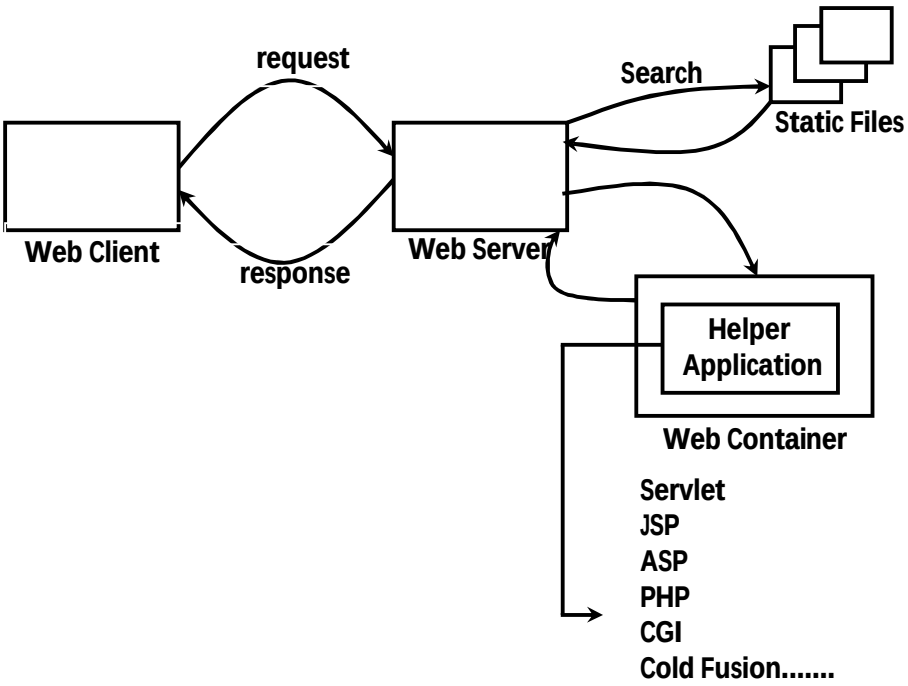
Web Programming for Static Response:



1. Client sends a request for static file to the server.
2. Server searches whether requested resource is available or not
3. If the requested resource is available then server will provide that file as response.
4.If the requested resource is not avaialble then we will get 404 status code saying requested resource is not avaialable.

Note: To serve static files , no processing is required at server side. Hence web server always Loves to serve static files.

Web programming for dynamic response:



client sends a request to the web server.
Web server will check whether the request is for static or for dynamic information based on url.

If it is for static information then web server will search for that static file.If it is available then server will provide that static file as response.If it is not available then server will provide 404 status code saying requested resource is not available.

If the request is for dynamic information then web server will forward the request to some helper application.

Helper application will analyze request,process request and generates required dynamic response.

Helper application forwards that response to web server and web server inturn forward that response to the client.

Hence to generate dynamic responses at server side some helper application is required.The following are various helper applications..
Servlet
JSP
ASP
PHP
CGI
COLD FUSION
etc..

Servlet:

Servlet is a server side webcomponent which is responsible to generate dynamic responses.
Total Servlet life cycle is managed by web container.

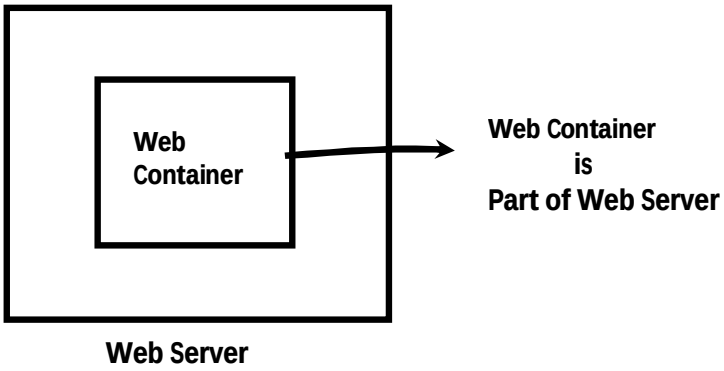
Note:
Web container is a big assistant to the programmer. It reduces the complexity of programming by managing total life cycle of servlet.

Types of web containers:

There are 3 types of web containers are possible.

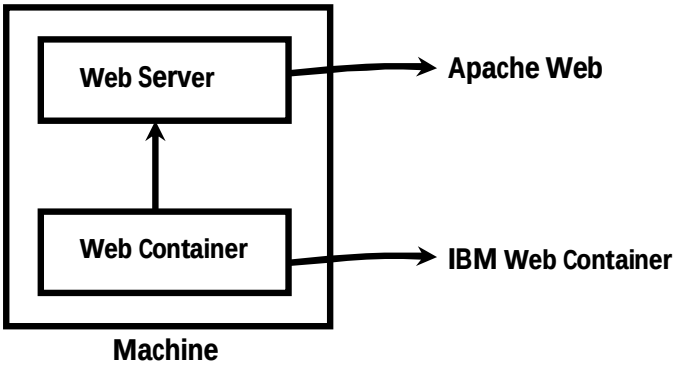
1. Stand Alone web containers
2. In process web containers
3. Out process web containers

1. Stand Alone web containers:



Both web server and web container are available as a single integrated component,such type of web containers are called stand alone web containers.
Eg: Tomcat

2. In Process Web Container:

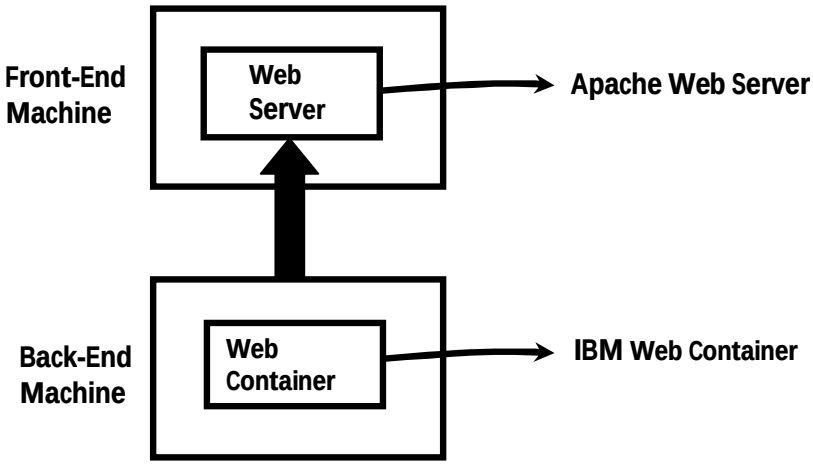


Web container is not part of web server.
web container is connected to the web server as plug-in and both web server and web container are running in the same address space(same machine),such type of web containers are called in process web containers.

The main advantage of in process web containers is both web server and web container need not be from the same vendor. We can use Apache web server and weblogic web container.

The type of web containers are suitable for small scale applications.

3. Out Process web containers:

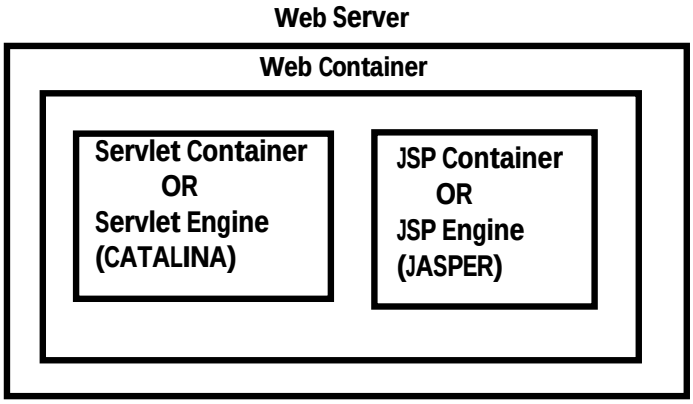


Web container is not part of web server.
Web server and web container are running on different machines .Web container is attached to the web server externally.

We can configure Front end Apache web server to forward request to the back end IBM web container.

This type of web containers are best suitable for real time applications.

Tomcat components:



Every web server internally contains web container.
web container is responsible to manage and execute servlets and jsps.

Internally web container contains two components

1. Servlet Container
2. JSP container

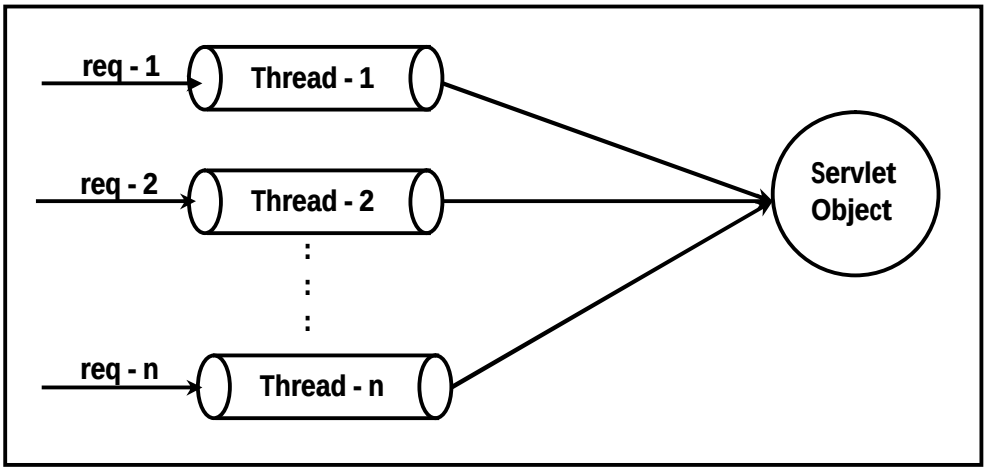
1. Servlet Container:

It is also known as Servlet Engine.
It is responsible to manage and execute Servlet components.
In Tomcat servlet container's name is CATALINA

2. JSP container:

It is also known as JSP Engine.
It is responsible to manage and execute JSP Components.
In Tomcat JSP Container's name is JASPER.

How Servlet Technology works:



For every Servlet, web container will creates only one object.
For every request, web container will allocate seperate thread,which is responsible to process that request.Hence servlet follows "Single Instance MultiThreaded Model".

Servlet API:

Servlet API defines the following 2 packages

1. javax.servlet package
2. javax.servlet.http package

1. javax.servlet:

This package defines several classes and interfaces used for developing protocol independent servlets(Generic Servlets)

2. javax.servlet.http :

This package defines several classes and interfaces which can be used to develop Http protocol based servlets.

Important interfaces of javax.servlet package:

1.Servlet
2.ServletRequest
3.ServletResponse
4.ServletConfig
5.ServletContext
6.RequestDispatcher
7.SingleThreadModel

1.Servlet (I):

Every Servlet in Java should implements Servlet Interface either directly OR indirectly.

This Interface defines the most common Methods which are applicable for any Servlet Object.

The Life Cycle Methods of Servlet are defined in this Interface only.

```
public class FirstServlet implements Servlet
{
}
```

2. ServletRequest(I):

For every request web container creates one request object.
ServletRequest object holds end user provided information.
Servlet can use this request object to get end user's provided information.

3. ServletResponse(I):

For every request web container creates one response object.
Servlet can use response object to prepare and send response to end user.

4.ServletConfig(I):

For every Servlet web container will creates a seperate config object to hold its configuration information.
Servlet can use this config object to get its configuration information.

5. ServletContext(I):

For every web application web container creates a seperate context object to hold application level configuration information.
Servlet can use this context object to get application level configuration information.

Note: ServletConfig is per Servlet where as ServletContext is per web application.

6.RequestDispatcher(I):

We can use RequestDispatcher to dispatch request from one servlet to another servlet.

7. SingleThreadModel(I):

Single servlet object can be accessed by multiple threads simultaneously and hence there may be a chance of data inconsistency problems. i.e servlet by default not thread safe.

To overcome this problem we should go for SingleTheadModel.

If our servlet class implements SingleThreadModel then our servlet object can be accessed by only one thread at a time.

```
public class FirstServlet implements Servlet,SingleThreadModel
{
}
```

The main advantage of SingleThreadModel is threads will be executed one by one and hence data inconsistency problems will be resolved.

But the main disadvantage of SingleThreadModel is, it increases waiting time of Threads and creates performance problems.Hence it is not recommended to use SingleThreadModel and it is deprecated in Servlet 2.4V.

Instead of SingleThreadModel it is recommended to use synchronized keyword.

SingleThreadModel interface does not contain any methods and it is marker interface.Internally JVM is responsible to provide required ability.

Important classes of javax.servlet package:

1.GenericServlet
2.ServletInputStream
3.ServletOutputStream
4.ServletException

1.GenericServlet:

GenericServlet implements Servlet interface.
GenericServlet acts as base class to develop protocol independent servlets.
Eg:

```
public class FirstServlet extends GenericServlet
{
}
```

2. ServletInputStream:

We can use ServletInputStream to read binary data send by end user.

3. ServletOutputStream:

We can use ServletOutputStream to write binary data to the response.

4.ServletException:

while processing our request if servlet faces any problem then we will get ServletException

Servlet(I):

Every servlet in java should implements Servlet interface either directly or indirectly.

Servlet interface defines the most common methods which are applicable for any servlet.

Servlet interface defines the following 5 methods

1.init()
2.service()
3.destroy()
4.getServletConfig()
5.getServletInfo()

1.init() method:

public void init(ServletConfig config) throws ServletException

This method will be called automatically by web container to perform initialization activities after servlet instantiation immediately.

Once init() method completes then only servlet is in a position to provide service.

2. service() method:

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

This method will be executed automatically by web container for every request to provide required response.

Total service logic,we have to write in this method only.

3.destroy() method:

public void destroy()

This method will be executed only once by the web container to perform cleanup activities just before taking servlet object from out of service.

Once destroy() method completes automatically webcontainer destroys that servlet object.

This is usually happens at the time of server shutdown or at the time of application undeployment.

Note: init(),service() and destroy() methods are called life cycle methods of servlet.

4.getServletConfig():

public ServletConfig getServletConfig()

This method returns ServletConfig object. By using this object servlet can get its configuration information.

5. getServletInfo()

public String getServletInfo()
This method returns information about our servlet like author,version,copyright information etc..

init()
service()
destroy()
getServletInfo()
getServletConfig()

Note:

init(),service() and destroy() methods are called callback methods because these methods will be executed automatically by the web container.

getServletConfig() and getServletInfo() methods should be called explicitly by the programmer based on our requirement and hence these methods are called in line methods.

Steps to develop First web application:

Step-1: Developing servlet by implementing Servlet interface:

Every Servlet in java should implements Servlet interface either directly or indirectly.

Whenever we are implementing Servlet interface compulsary we should provide implementation for all 5 methods of Servlet interface.

FirstServlet.java:

```
1)  import javax.servlet.*;
2)  import java.io.*;
3)  import java.util.*;
```

```
4)  public class FirstServlet implements Servlet
5)  {
6)     static
7)     {
8)        System.out.println("servlet class loading...");
9)     }
10)    public FirstServlet()
11)    {
12)       System.out.println("servlet instantiation...");
13)    }
14)    public void init(ServletConfig config) throws ServletException
15)    {
16)      System.out.println("init() method execution...");
17)    }
18)    public void service(ServletRequest req,ServletResponse resp) throws ServletException,I
    OException
19)    {
20)       resp.setContentType("text/html");
21)       System.out.println("service() method execution...");
22)       PrintWriter out=resp.getWriter();
23)       out.println("<h1>Welcome Innocent Advanced Java Students</h1>");
24)       out.println("<h1>The Server Time is:"+new Date()+"</h1>");
25)    }
26)    public void destroy()
27)    {
28)       System.out.println("destroy() method execution...");
29)    }
30)    public ServletConfig getServletConfig()
31)    {
32)       return null;
33)    }
34)    public String getServletInfo()
35)    {
36)       return "Developed by Durga";
37)    }
38) }
```

The basic purpose of servlet is to provide response to the end user.

We can set content type of response as follows

resp.setContentType("text/html");

MIME Type represents the type of response we are sending to the end user.

The default MIME Type is : text/html

Other popular MIME Types are:
application/pdf
image/jpeg
video/mp4
etc..

We can write text data(character data) to the response by using PrintWriter.

We can get PrintWriter which is pointing to response as follows...

PrintWriter out=resp.getWriter();

By using this PrintWriter if we are writing any text data ,it will be written to response object,which will be delivered to the end user.

By using SOP() statements if we are writing anything,it will be displayed to the server console and won't send to the end user.

Note:

Servlet Programs won't be run by programmer and web server is responsible to run. Hence main method concept is not applicable to servlet classes and we cannot run servlet from command prompt.

Step 2: Compilation of Servlet class:

After developing servlet class, we have to compile just like our normal java class.
In the servlet program what ever dependent classes we used, are available in servlet-api.jar.

web server vendor is responsible to provide this jar file.

In Tomcat installation this jar file is available in the following location

D:\Tomcat 7.0\lib
Hence to compile a servlet, we have to place this jar file in the classpath.

classpath
D:\Tomcat 7.0\lib\servlet-api.jar

Step-3: Creation of Deployment Descriptor(web.xml):

For every web application we have to provide one xml file named with web.xml. This web.xml file is also known as Deployment Descriptor.

web container will use this xml file to get information about our servlets.Hence web.xml acts as guide to web container.

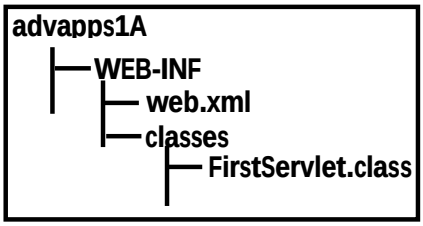
```
1)  <web-app>
2)    <servlet>
```

```
3)      <servlet-name>DemoServlet</servlet-name>
4)      <servlet-class>FirstServlet</servlet-class>
5)    </servlet>
6)
7)    <servlet-mapping>
8)       <servlet-name>DemoServlet</servlet-name>
9)     <url-pattern>/test</url-pattern>
10)   </servlet-mapping>
11) </web-app>
```

Step-4: creation of web application folder structure:

Servlet API defines some standard structure for web application

Every web server can able to understand this application structure.



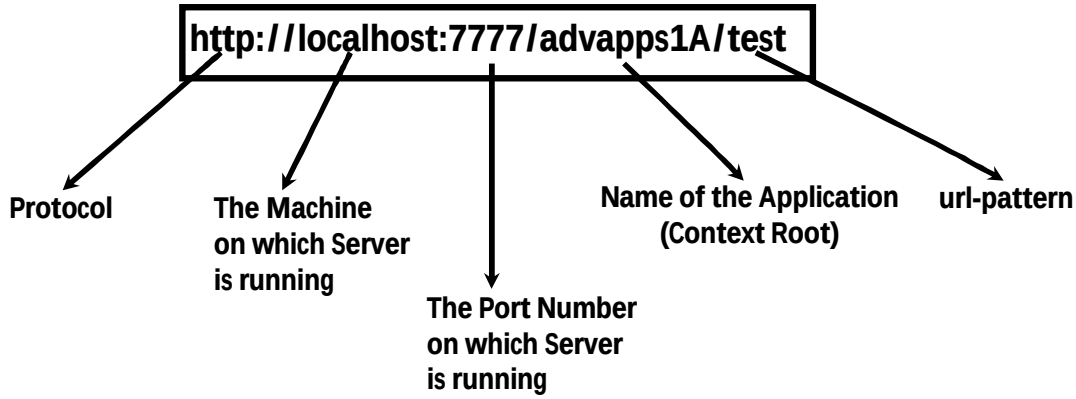
Step-5: Deployment of web application in the web server:

Once we developed web application according standard folder structure, we have to place this application inside web server. This process is called deployment.

We have to place our application in the Tomcat Server of following location.(D:\Tomcat 7.0\webapps)

Step-6: start server and send the request

Once we deployed application in the web server,we can send the request as follows...



For the First Request:

servlet loading
Servlet Instantiation
Init Method called
Service Method called

For the Second Request onwards:

Service Method called

Note: At the time of first request servlet class will be loaded and servlet object will be created followed by init() method execution.Finally service() method will be called.
But for second request onwards only service() method will be called.
Because of this the processing time of first request is more when compared with other requests.

To overcome this problem we should go for <load-on-startup>. If we configured <load-on-startup> then servlet class loading,servlet instantiation and execution of init() method will be performed at the time of server start up or at the time of application deployment.
For the first request also only service() method will be called.

We can configure <load-on-startup> in web.xml as follows...

```
<web-app>
        <servlet>
          ...
          <load-on-startup>10</load-on-startup>
        </servlet>
        .....
 </web-app>
```

The main advantage of <load-on-startup> is all requests will be processed with uniform response time.

Case 1: without <load-on-startup>:

First Request:

servlet loading
servlet instantiation
init() method execution
service() method

Second Request onwards:
service() method

Case 2: with <load-on-startup>:

At the time of server start up or at application deployment:

servlet loading
servlet instantiation
init() method execution

First Request:
service() method called

Second Request:
service() method called

Without <load-on-startup>	With <load-on-startup>
First Request: Servlet Loading Servlet Instantiation init() Method Execution service() Method Execution **Second Request Onwards:** service() Method Execution	**At The Time Of Server Start Up OR At Application Deployment** Servlet Loading Servlet Instantiation init() Method Execution **First Request:** service() Method Execution **Second Request:** service() Method Execution

Note: From Servlet 2.5 Version onwards we can define multiple url-patterns for the same servlet in web.xml as follows...

```
<web-app>
      .....
            <servlet-mapping>
              <servlet-name>DemoServlet</servlet-name>
              <url-pattern>/test</url-pattern>
              <url-pattern>/demo</url-pattern>
              <url-pattern>/hello</url-pattern>
            </servlet-mapping>
 </web-app>
```

http://localhost:7777/advapps1A/test
http://localhost:7777/advapps1A/demo
http://localhost:7777/advapps1A/hello

Note:
At the time of application undeployment or at the time of server shutdown destroy() method will be executed.

Life Cycle of Servlet that implements Servlet interface:

1. Client(Browser) sends a request to the server.

2. Server will check whether the request is for static or for dynamic information based on url

3.If the request is for static information, then the server will search for the required static file.If the static file is available then web server provides that static file as response.If the required static file is not available then server will send 404 status code saying requested resource is not available.

4.If the request is for dynamic information then web server forwards the request to the web container.

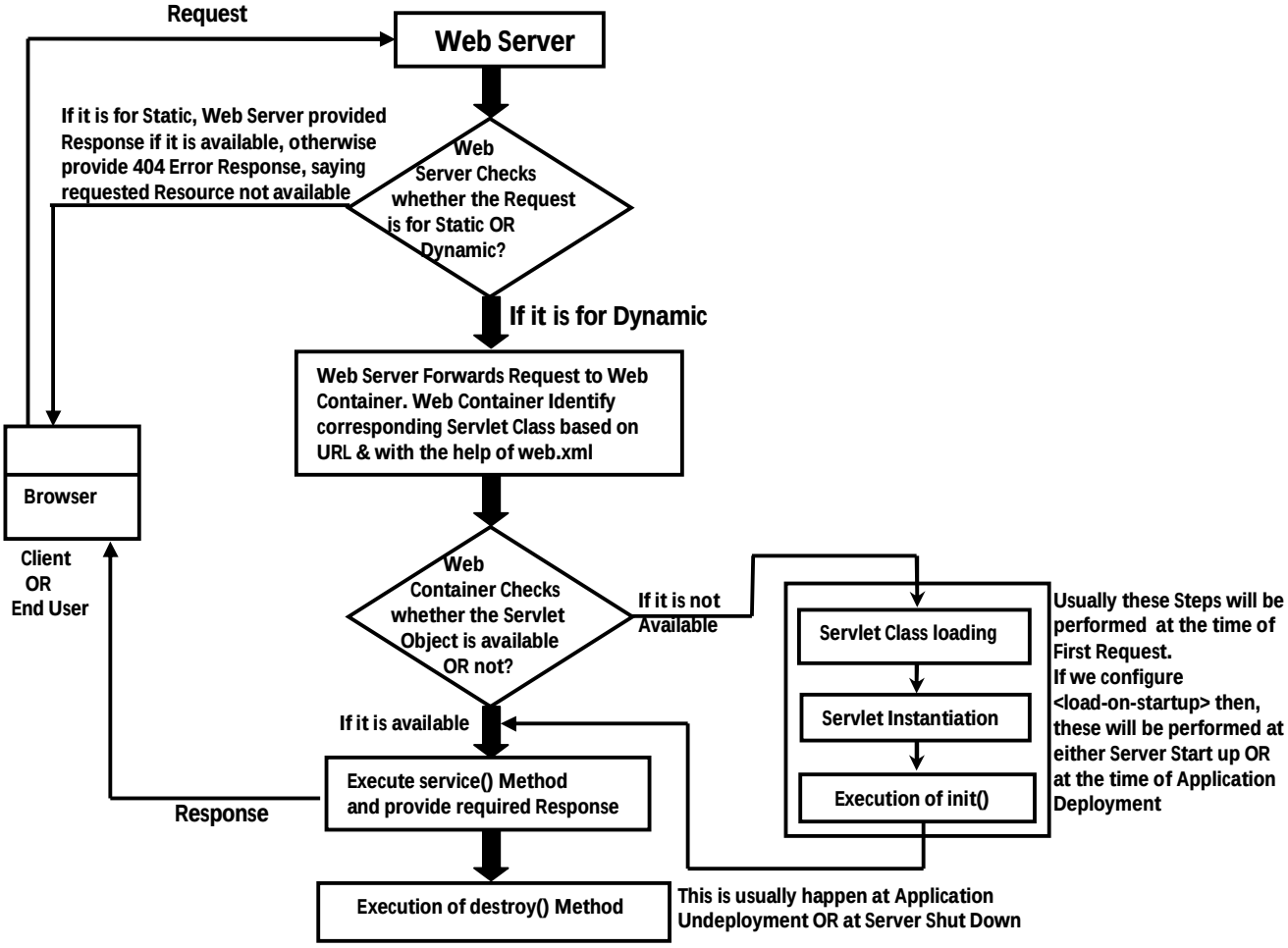
5.Web container will identify the corresponding servlet class based on url pattern and with the help of web.xml

6.web container will check whether servlet object is available or not
If it is not already available,then web container will load servlet class,and create object for that servlet class and execute init() method.
servlet class loading(by Class.forName() method)
Servlet Instantiation(by uisng newInstance() method)

7. web container will call service() method which is responsible to provide required response.

8. Finally web container will call destroy() method to perform cleanup activities. This is usually happend at the time of application undeployment or at the time of server shutdown.

Flow Chart for Servlet Life Cycle that implements Servlet interface:



Developing Annotation based Servlet in Servlet 3.0V:

In Servlets Annotations concept introduced in 3.0V. Annotation means MetaData(Data about Data) i.e Annotation provides extra information about our components.
All servlet related annotations are avaialble in the following package.
javax.servlet.annotation

Whenever we are using annotations, compulsary we should import the above package.

import javax.servlet.annotation.*;

up to certain level we can replace web.xml configurations with annotations. Some times we can remove web.xml file also.

We can define url-pattern by using annotation as follows @WebServlet("/test")

We can define multiple url patterns as follows @WebServlet({"/test","/demo","/hello"})

FirstServlet.java:

```
1)  import javax.servlet.*;
2)  import java.io.*;
3)  import java.util.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test")
6)  public class FirstSevlet implements Servlet
7)  {
8)     static
9)     {
10)       System.out.println("servlet class loading..");
11)    }
12)    public FirstSevlet()
13)    {
14)       System.out.println("Servlet class Instantiation...");
15)    }
16)    public void init(ServletConfig conf) throws ServletException
17)    {
18)       System.out.println("Init Method called");
19)    }
20)    public void service(ServletRequest req , ServletResponse resp) throws ServletException,I
    OException
21)    {
22)       System.out.println("Service Method called");
23)       resp.setContentType("text/html");
24)       PrintWriter out = resp.getWriter();
25)       out.println("<html><body bgcolor=green text=white><h1>Welcome Innocent Adv.Jav
    a Students<br/>");
26)       out.println("The Server Time is :"+new Date()+"</h1></body></html>");
27)    }
28)    public void destroy()
29)    {
30)       System.out.println("Destroy Method called");
31)    }
32)    public ServletConfig getServletConfig()
33)    {
34)       return null;
35)    }
36)    public String getServletInfo()
37)    {
38)       return "written by durga";
39)    }
40) }
```

http://localhost:7777/advapps1A3X/test

Annotation Based Servlet:

```
import javax.servlet.annotation.*;
@WebServlet("/test")
public class FirstSevlet implements Servlet
{
  ...
}
```

```
eg2:
import javax.servlet.annotation.*;
@WebServlet({"/test","/hello","/demo"})
public class FirstSevlet implements Servlet
{
 ...
}
```

Servlet 2.x Based Folder Structure:

```
advapps1A
  |-- WEB-INF
     |-- web.xml
     |-- classes
        |--FirstServlet.class
```

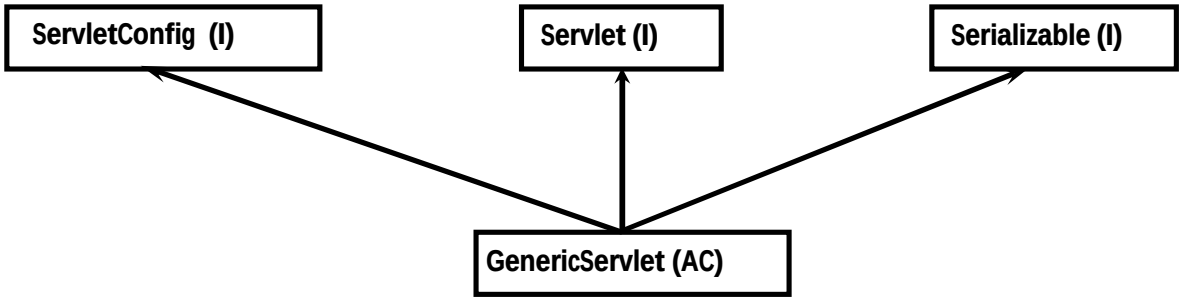
Servlet 3.x Based Folder Structure:

```
advapps1A
  |-- WEB-INF
     |-- classes
        |--FirstServlet.class
```

GenericServlet:

We can develop a Servlet by implementing Servlet interface directly. In this case compulsary we should provide implementation for all 5 methods of Servlet interface whether required or not required. Because of this length of the code will be increases and reduces readability.

To overcome this problem we should go for GenericServlet.
GenericServlet already implemented Servlet interface and provides implementation for all methods of Servlet interface except service() method. Instead of implementing Servlet interface if we extend GenericServlet then we required to provide implementation only for service() method, so that length of the code will be reduced and readability will be improved.



GenericServlet is an abstract class and implements Servlet,ServletConfig and Serializable interfaces...

Developing Servlet by extending GenericServlet:

FirstServlet.java:

```
1)  import javax.servlet.*;
2)  import java.io.*;
3)  import javax.servlet.annotation.*;
4)  @WebServlet("/test")
5)  public class FirstSevlet extends GenericServlet
6)  {
7)     public void service(ServletRequest req, ServletResponse resp) throws ServletException,I
    OException
8)     {
9)        PrintWriter out = resp.getWriter();
10)       out.println("<h1>Writing servlet by extending GS is very easy</h1>");
11)    }
12) }
```

```
advapps1B
  |-- WEB-INF
     |-- classes
        |--FirstServlet.class
```

Note:

In the above program two classes are avaialble FirstServlet and GenericServlet. Web container creates object for FirstServlet class.

If web container calls any method , first it will check whether our FirstServlet class contains that method or not. If our class contains that method then it will be executed. If our class does not contain that method then only parent class method(GenericServlet) will be executed.

First priority for our class and then GenericServlet class because object is available for our class.

Internal implementation of GenericServlet:

```
1)  public abstract class GenericServlet implements Servlet, ServletConfig, Serializable
2)  {
3)     private transient ServletConfig config;
4)
5)     public void init(ServletConfig config) throws ServletException
6)     {
7)        this.config = config;
8)        init();
9)     }
10)
11)    public void init() throws ServletException
```

```
12)   {
13)   }
14)
15)   public ServletConfig getServletConfig()
16)   {
17)      return config;
18)   }
19)  ::::::::::::::::::::::::::::::::
20) }
```

Case-1: If our servlet class does not contain any init() method:

If our servlet class does not contain any init() method then web container always calls init(SC config) method of GenericServlet.
Inside this method config object will be saved for the future purpose and calls no-arg init() method.
web container will check whether our servlet class contains no-arg init() method or not.If our servlet class does not contain no-arg init() method, then GenericServlet no-arg init() method will be called, which has empty implementation.

In this case getServletConfig() method returns config object.

CASE 2: If we override init(SC) in our servlet class:

```
public void init(ServletConfig config)throws SE
{
  SOP("initialization activities");
}
```

Web container will always calls our class init(SC) method. This is way of overriding init() method is not recommended b'z we are not saving config object for the future purpose. In this case getServletConfig() method returns null.

Case-3: If we override no-arg init() method:

```
public void init()throws SE
{
  SOP("initialization activities");
}
```

In this case web container will always calls GenericServlet class init(SC) method, which saves config object for the future purpose and internally calls no-arg init() method.Then our servlet class no-arg init() method will be called.

In this case getServletConfig() method returns config object.

Note:
1.If our servlet class does not contain any init()method:
GS: init(SC)===>GS:init()

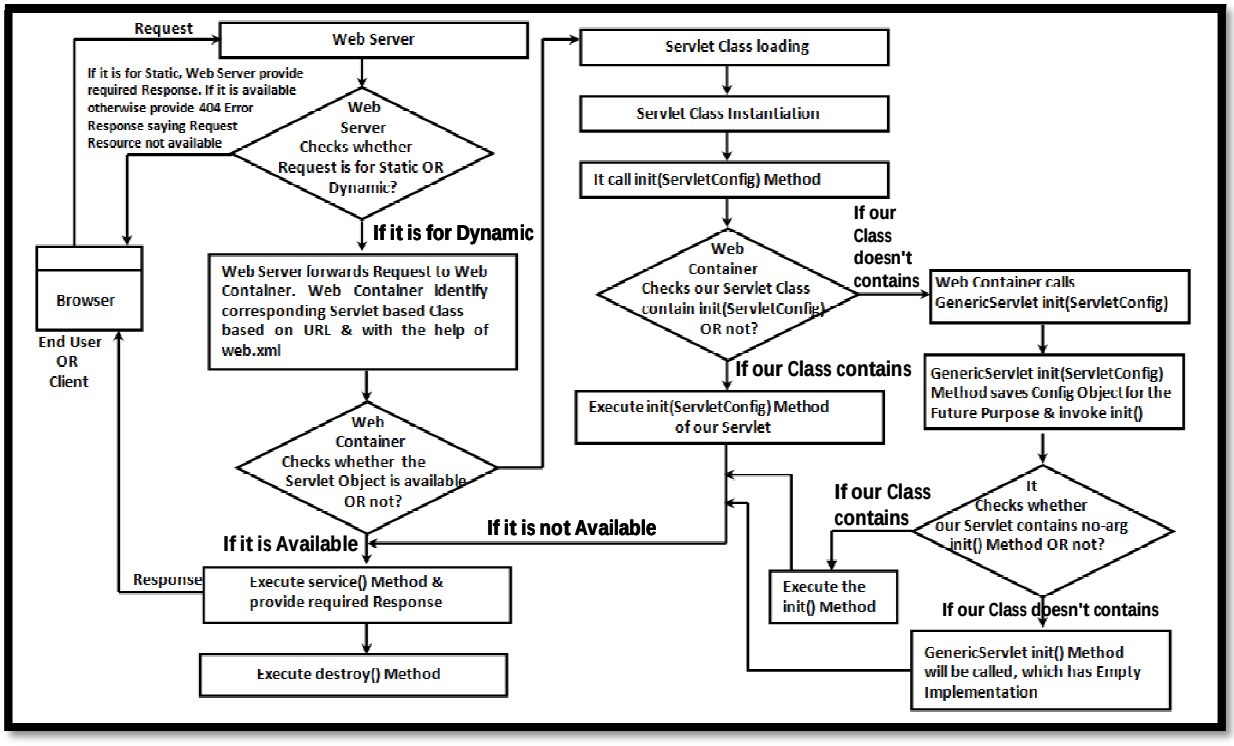
2.If our servlet class contains init(SC) method
FS:init(SC)
3.If our servlet class contains no-arg init() method:
GS:init(SC) ===>FS:init()

Q.why GenericServlet class contains 2 init() methods?
init(SC) is for web container purpose
init() for programmer purpose

Q.In our servlet which init() method is recommended to override?
no-arg init() method

Q.In GenericServlet class config variable declared as transient. what is the reason?
due to security constraints config object should not be travelled across the network.

Flow Chart For Servlet Life Cycle That extends GenericServlet



javax.servlet.http:
This package contains several classes and interfaces which can be used for developing Http based servlets.

Important interface of javax.servlet.http package:

1.HttpServletRequest:
It is the child interface of ServletRequest

We can use request object to get end user provided information.

2.HttpServletResponse:
It is the child interface of ServletResponse
We can use response object to prepare and send response to the end user.

3.HttpSession:
We can use HttpSession object to implement session management.

Important classes of javax.servlet.http package:

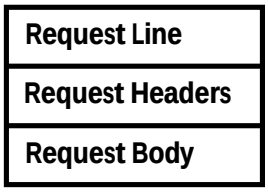
1.HttpServlet:
It is the child class of GenericServlet.
HttpServlet acts as base class to develop Http based servlets.

2. Cookie:
We can use Cookies in the session management.

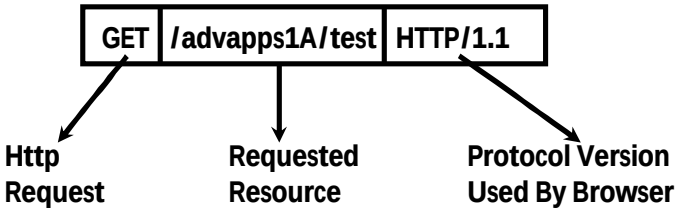
Common Terminology:
web client and web server communicates with each other by using some common language known as HTTP.

HTTP defines standard structures for request and response.

Structure of Http request:
Request Line
Request Headers
Request Body



Request Line:



Request Headers:
Request headers will provide configuration information of the browser like media types accepted by browser,languages accepted by browser,encoding types supported by browser etc..

Web server will use these request headers while preparing proper response.

Request Body:

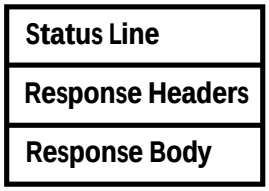
It contains end user provided information like resume,username etc...

Note:

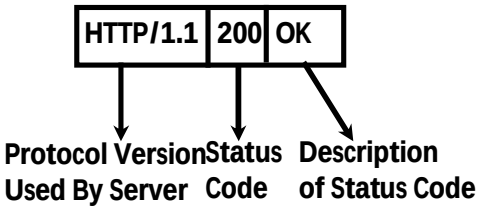
For the GET request, request body is optional and for POST request, request body is mandatory.

Structure of Http Response:

Status Line
Response Headers
Response Body



Status Line:



1XX==>Informational
2XX===>Successful
3XX===>Redirection
4XX===>Client Error
5XX===>Server Error

Response Headers:

These provide configuration information of server and information about our response like content type, content length,last modified date etc..
Browser will use these response headers to display response properly to the end user.

Response Body:

It contains original response provided by server.This response body will be displayed to the end user.

Note:

Request headers send by browser and used by server where as Response headers send by server and used by browser.

Types of HTTP Request Methods:

Based on operations expected by client all http methods are divided as follows...

- **Big-7 HTTP Methods**
  - 1. GET
  - 2. POST
  - 3. HEAD

  } Introduced in HTTP/1.0 V
  - 4. PUT
  - 5. DELETE
  - 6. OPTIONS
  - 7. TRACE

  } Introduced in HTTP/1.1 V
- 8. CONNECT
- 9. MOVE
- 10. LOCK
- ::::::::::::::

GET,POST and HEAD introduced in HTTP/1.0V

The first 7 methods are called Big-7 Http Methods

Note:

Until Servlet 2.4V,web server can provide support for the first-7 HTTP methods.But from Servlet 2.5 version onwards web server can provide support for the remaining methods also.

Note:

Being java developer we have to aware only GET and POST methods.

GET Method:

1. If we want to get information from the server then we should go for GET Request.

2. Usually GET Requests are Read Only and at server side update operation won't be performed.

3.In GET Request end user's provided information will be appended to the url as the part of Query String.

Eg: http://localhost:7777/advapps1C/test?uname=durga&pwd=java

4. In the GET request end user's provided information is visible to the outside and hence there may be a chance of security problems.B'z of this, we cannot send sensitive information(like username,pwd etc) by using GET Request.

5.We can send only limited amount of information by GET Request.

6. We can send only character data(text data) and we cannot send binary data(like images,video files etc) by using GET Request.

7. Book marking of GET Request is possible.

8. Caching of GET Request is possible and hence performance will be improved.

Idempotent Request:

By repeating the request multiple times if there is no change in the response,such type of requests are called Idempotent Requests.

Eg: GET is idempotent where as POST is not idempotent

Safe Request:

By repeating the request several times ,if there is no side effect at server side,such type of requests are called safe requests.
Eg: GET is safe but POST is not safe

Triggers to send GET Request:

1. Typing url in the address bar and submit
2. Clicking hyperlink
3. submit the form without method attribute
4. submit the form with method attribute value as GET.

```
<form action="/test"  method="GET">
  .....
</form>
```

POST Method:

1. We can use POST method to send huge amount of information to the server
Eg: uploading resume
2. usually POST requests are write only. i.e at server side update operation will be performed.
3.For the POST requests end user provided information will be encapsulated in the request body instead of appending to the url.
4.End users data is not visible in the url and hence we can send sensitive informtion by using POST request.
5.There is no limit to the size of request body and hence we can send huge amount of information by using POST request.
6.By using POST request we can send both character and binary data.
7.Book marking of POST Request is not possible
8.Caching is not applicable for POST requests
9. Post requests are not idempotent
10. Post requests are not safe

Triggers to send POST Requests:
There is only one way to send POST Request. i.e submit the form with method attribute of POST value.

```
<form action="/test1" method="POST">
  ...
</form>
```

Note: For GET requests body is optional where as for POST Requests body is mandatory.

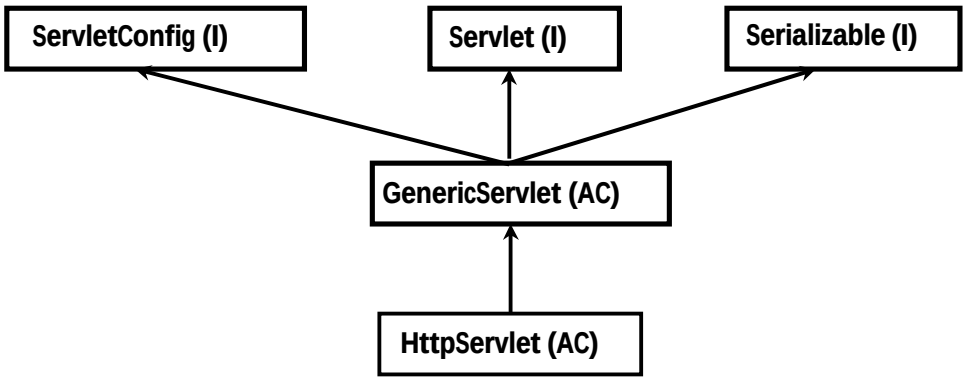
Q. What are differences b/w GET and POST?

GET	POST
1) We can use GET Method to GET Information from the Server.	1) We can use POST Method to POST Information to the Server.
2) Usually GET requests are READ-ONLY.	2) Usually POST Requests are WRITE or UPDATE Operations.
3) End User provided Information will be append to the URL as the Part of Query String and send to the Server.	3) End User provided Information will be encapsulated in the Request Body and send to the Server.
4) By using GET Request we can send only Character Data (ASCII) and we cannot send Binary Data like Images.	4) By using POST Request we can send both Binary and Character Data to the Server.
5) By using GET Request we can send only limited Amount of Information, which is varied from Browser to Browser.	5) By using POST Request we can send huge Amount of Information to the Server.
6) Security is less and hence we cannot send sensitive Information like User Name, Password etc.	6) Security is more and hence we can send sensitive Information like User Name, Password etc.
7) Book Marking of GET Request is possible.	7) Book Marking of POST Request is not possible.
8) Caching of GET Request is possible.	8) Caching of POST Request is not possible.
9) GET Request is Idempotent.	9) POST Request is not Idempotent.
10) GET Request is safe.	10) POST Request is not safe.
11) There are multiple ways to send GET Request: • Typing URL in the Address Bar and enter • Clicking Hyper Link • Submitting the HTML FORM with Method Attribute of GET Value.	11) There is only one way to send POST Request: • Submitting the HTML FORM with Method Attribute of POST Value.

HttpServlet:

HttpServlet acts as base class to develop Http Based servlets.

It is the child class of GenericServlet.



For every Http Method XXX,HttpServlet class contains the corresponding doXxx() method.

protected void doXxx(HttpServletRequest req,HttpServletResponse resp)throws ServletException,IOException

Eg:
protected void doGet(HttpServletRequest req,HttpServletResponse resp)throws ServletException,IOException

HttpServlet contains 2 Service methods

1. public void service(ServletRequest req,ServletResponse resp)throws SE,IOE

2. protected void service(HttpServletRequest req,HttpServletResponse resp)throws SE,IOE

Demo Program for developing servlet by extending HttpServlet:

login.html:(we can use to send POST request)

```
1)  <html>
2)  <body><h1> This is HttpServletDemo to send Post request</h1>
3)  <form action = "/advapps1C/test" method="POST">
4)  Enter Name :<input type="text" name="uname"><br/>
5)  <input type="submit" value="Login">
6)  </form>
7)  </body>
8)  </html>
```

FirstServlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test")
6)  public class FirstSevlet extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
9)     {
10)       PrintWriter out = resp.getWriter();
11)       out.println("<h1>This is from doGET()method...</h1>");
12)    }
13)    public void doPost(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
14)    {
15)       PrintWriter out = resp.getWriter();
16)       out.println("<h1>This is from doPOST()method...</h1>");
```

```
17)    }
18) }
```

advapps1C
|-- login.html
|-- WEB-INF
|-- classes
|--FirstServlet.class

If we are sending GET request then doGet() method will be executed and if we are sending POST request then doPost() method will be executed.

Life cycle of HttpServlet:

1. Whenever we submit form ,browser prepares HttpRequest and send to the server.

2.Web server checks whether request is for static or for dynamic information.

3.If the request is for static information then web server provides the required response if it is available,otherwise it will send 404 status code saying requested resource is not available.

4.If the request is for dynamic information then web server forwards the request to web container.

5.web container will identify corresponding servlet class based on url pattern and with the help of web.xml

6.web container will check whether servlet object is available or not.

7.If the servlet object is not available then web container loads servlet class,instantiate servlet and execute init() method.

[execution of init() method is exactly same as GenericServlet life cycle]

8.web container creates ServletRequest and ServletResponse objects and invokes public service() method by passing these as arguments.
9.web container will check whether our servlet class contains public service(SR,SR) method or not. If our servlet class contains public service(SR,SR) method then it will be executed and provides required response.

10.If our servlet class does not contain public service() method then parent class(HttpServlet) public service(SR,SR) method will be executed, which is implemented as follows...

```
public void service(SR req, SR resp) throws SE, IOE
{
    HttpServletRequest request=(HSR)req;
    HttpServletResponse response=(HSR)resp;
    service(request, response);
}
```

Inside HttpServlet public service() method, req and resp objects will be type casted to HttpServletRequest and HttpServletResponse and then invoke protected service(HSR,HSR) method.
web container will check whether our servlet class contains protected service(HSR,HSR) method or not.
If our servlet class contains protected service(HSR,HSR) method then it will be executed and provide required response to the end user.
If our servlet class does not contain protected service() method then web container will execute HttpServlet protected service() method.
HttpServlet protected service() method will identify request method(like get,post etc) and invoke corresponding doXxx() method.

```
protected void service(HSR req,HSR resp)throws SE,IOE
{
      String method = req.getMethod();
      if(method.equals("GET"))
      {
         doGet(req,resp);
      } else
      if(method.equals("POST"))
      {
         doPost(req, resp);
      }
                     ...
          else
      {
            return 501 status code saying http method not implemented.
      }
}
```

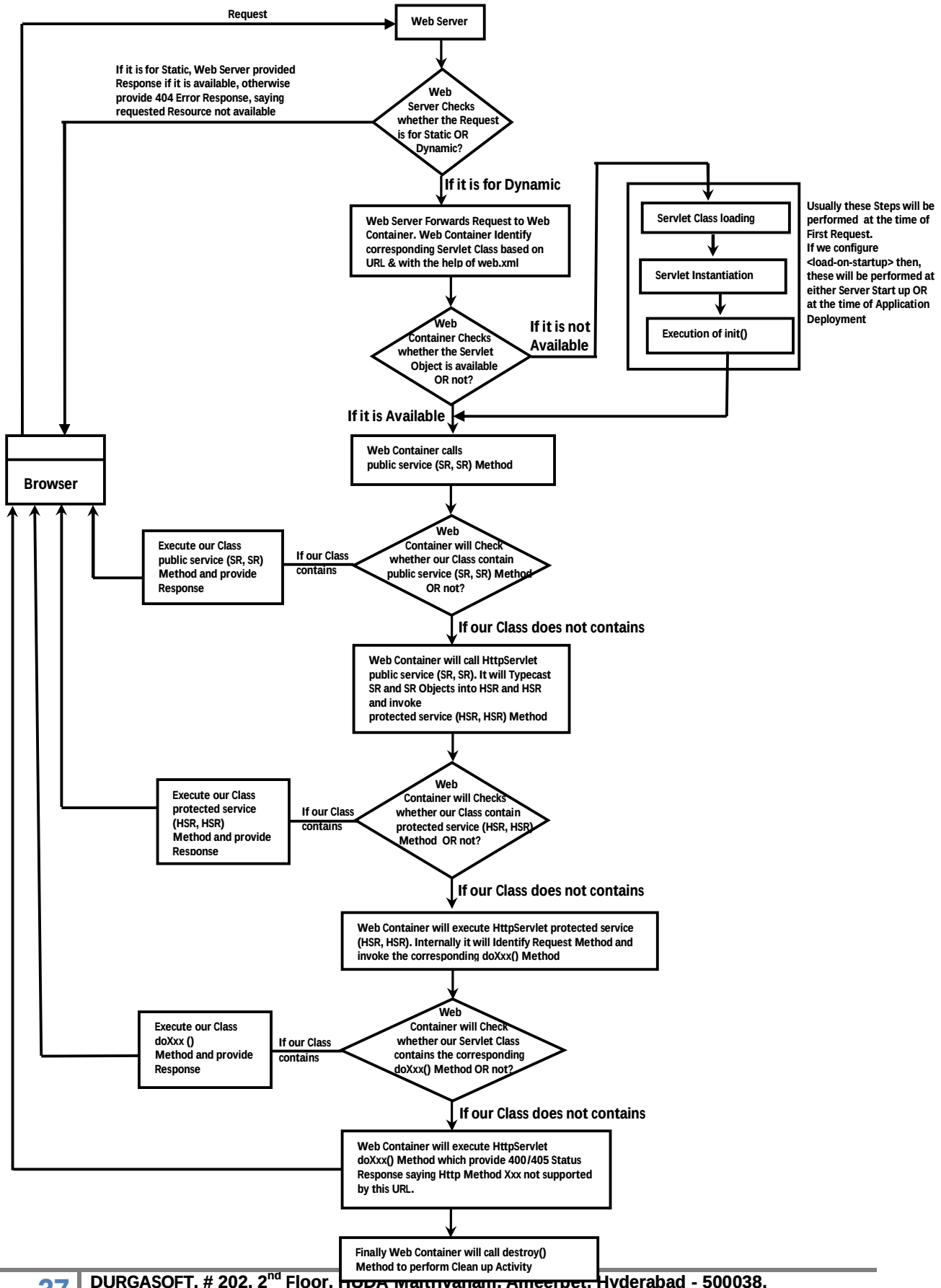
webcontainer will check whether our servlet class contains the corresponding doXxx() method or not.If our servlet class contains doXxx() method then it will be executed and provide required response.
If our servlet class does not contain doXxx() method then parent class HttpServlet doXxx() method will be executed,which is implemented as follows...

```
protected void doGet(HSR req,HSR resp)throws SE,IOE
{
      return 405|400 status code saying Http Method GET is not supported by this url.
}
```

HttpServlet doXxx() method wont do anything and just it will share error information to the end user.

Note:
Web Container will always calls the methods in the following order
public service(SR,SR)
protected service(HSR,HSR)
public doXxx(HSR,HSR)

Flow Chart For Servlet Life Cycle That extends HttpServlet



DURGASOFT, # 202, 2nd Floor, HUDA Maithrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Case-1:
If our servlet class contains public service(SR,SR) method then for any type of request(like get,post etc)
only this public service(SR,SR) method will be executed.

case-2:
If our servlet class contains both public service(SR,SR) and protected service(HSR,HSR) then public service(SR,SR) method will be executed for any type of request(like get,post etc)

case-3:
If our servlet class contains protected service() and doGet() methods,then for any type of request including GET, protected service() method will be executed.

*****case-4:**
If we are sending GET request but our servlet does not contain doGet() method and it contains doPost() method,then HttpServlet doGet() method will be executed which returns error information

HTTP Status 405 - HTTP method GET is not supported by this URL

case-5:
If we are sending POST request,but our servlet class does not contain doPost() method and it contains doGet() method then HttpServlet doPost() method will be executed which returns error information

HTTP Status 405 - HTTP method POST is not supported by this URL

case-6:
To provide common response for both GET and POST requests,we have to implement our servlet as follows..

```
1)  public class FirstSevlet extends HttpServlet
2)  {
3)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
4)     {
5)        PrintWriter out = resp.getWriter();
6)        out.println("<h1>This is common response for both GET and Post methods...</h1>");
7)     }
8)     public void doPost(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
9)     {
10)       doGet(req,resp);
11)    }
12) }
```

FAQS:

1. Why GenericServlet class is declared as abstract?
2. HttpServlet class does not contain any asbtract method, still it is declared as abstract.What is the reason?
3. Internally how web container will create servlet object?
4.Why every servlet should contains public no-arg constructor?
5.The main purpose of constructor is to perform initilization and every servlet compulsary contains public no-arg constructor.Then what is need of init() method?
6.Is it possible to call destroy() method explicitly from init() and service() methods?
7. In How many ways we can develop servlet?
8. In Http based servlets , is it recommended to override service() method?

Q1. Why GenericServlet class is declared as abstract?
GenericServlet implements Servlet interface and provides implementation for all methods except service() method.Hence GenericServlet class is declared as abstract.

Q2. HttpServlet class does not contain any asbtract method, still it is declared as abstract.What is the reason?

Most of the methods present inside HttpServlet class are implemented to send error information to the end user.B'z of this partial implementation HttpServlet class is declared as abstract.

Note:
1. If a class contains at least one abstract method then compulsary we should declare that class as abstract otherwise we will get compile time error.
eg: GenericServlet

2. Eventhough class does not contain any abstract method still class declared as abstract if implementation is partial.i.e abstract class can contain zero number of abstract methods also.
eg: HttpServlet

Q3. Internally how web container will create servlet object?

If we know class name at the beginning then we can create object by using new operator.
If we don't know class name at the beginning and it is available dynamically at runtime then we can create object by using newInstance() method.

Web container don't know our servlet class names at the beginning and hence it is always using newInstance() method to create Servlet object.

Q4.Why every servlet should contains public no-arg constructor?

web container will create servlet object by using newInstance() method.Internally newInstance() method will call public no-arg constructor. Hence every servlet class should compulsary contain public no-arg constructor,otherwise we will get java.lang.InstantiationException.

5. The main purpose of constructor is to perform initilization and every servlet compulsary contains public no-arg constructor.Then what is need of init() method?

To perform initialization of servlet compulsary we should pass ServletConfig argument.But in servlets constructor cannot take any arguments b'z newInstance() method.
Hence we cannot use constructor concept to perform initialization of the servlet.
Some mechanism must be required to perform initialization which is nothing but init() method.

init()==>can take ServletConfig as argument
Constructor==>cannot take any argument

Q6. Is it possible to call destroy() method explicitly from init() and service() methods?

Just before destroying servlet object web container will always calls destroy() method to perform clean up activities.

But based on our requirement we can call destroy() method explicitly from init() and service() methods,then it will be executed just like a normal method and servlet object wont be destroyed.

```
init()
{
   destroy();
}
 service()
 {
  destroy();
 }
```

Q7. In how many ways we can develop a servlet?

3 ways
By implementing Servlet interface directly
By extending GenericServlet
By extending HttpServlet

Q8. In Http based servlets , is it recommended to override service() method?
No. If we override service() method then for any type of request same service() method will be executed.

Playing Games with Request object

Objective:

By using HttpServletRequest, write code for

1. Retrieving form parameters
2. Retrieving Request headers
3. Retrieving Cookies
4. Retrieving Client and Server information

Retrieving Form Parameters:

Form Parameters are key-value pairs where both key and value are String objects.
Form parameter can be associated with single value or multiple values.
ServletRequest interface defines the following methods to retrieve form parameters from the request object.

1.public String getParameter(String pname)

returns the value associated with specified parameter
If parameter associated with multiple values then it returns only first value.
If the specified parameter is not available then this method returns null.
Argument is case sensitive.
Eg: String user=req.getParameter("uname");

2. public String[] getParameterValues(String pname)

It returns all values associated with the specified parameter.
If the specified parameter is not available then this method returns null.
Argument is case sensitive

```
Eg: String[] s = req.getParameterValues("course");
    for(String s1: s)
    {
       SOP(s1);
    }
```

3. public Enumeration getParmeterNames()

returns all form parameter names.

4. public Map getParameterMap()

returns Map object which contains all parameter names and values.

getParameter()
getParameterValues()
getParameterNames()
getParameterMap()

Demo Program to retrieve Form Parameters:

register.html:

```
1)  <html>
2)  <body bgcolor=green text=white><center><h1>Durga Software Solutions:Registration for
    m</h1></center>
3)  <form action = "/advapps1D/test" method="GET">
4)  <table border=0>
5)  <tr><td>Enter Name :</td><td><input type=text name=uname></td></tr>
6)  <tr><td>Enter contact number:</td><td>
7)  <input type=text name=ucontact></td></tr>
8)  <tr><td>Enter Course:</td><td>
9)  <select name=course multiple>
10) <option value=CoreJava>Core Java</option>
11) <option value=AdvJava>Advanced Java</option>
12) <option value=Hibernate>HIBERNATE</option>
13) <option value=Spring>SPRING</option>
14) <option value=WebServices>Web Services</option>
15) </select></td></td>
16) <tr><td><input type=submit></td><td></td></tr></table>
17) </form>
18) </body>
19) </html>
```

FormDemoServlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test")
6)  public class FormDemoServlet extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
9)     {
10)       PrintWriter out = resp.getWriter();
11)       String username=req.getParameter("uname");
12)       String usercontact=req.getParameter("ucontact");
13)       out.println("<h1>User Name:"+username+"<br>");
14)       out.println("User Contact:"+usercontact+"<br>");
15)       String[] s = req.getParameterValues("course");
16)       out.println("Your Selected Courses:<br/>");
17)       for(String s1 : s)
18)       {
19)          out.println(s1+"<br>");
20)       }
21)       out.println("</h1>");
```

```
22)    }
23)
24) }
```

```
advapps1D
   |-register.html
   |-WEB-INF
      |-classes
            |-FormDemoServlet.class
```

Retrieving Request Headers:

For every request browser sends its configuration information in the form of headers. These may include the media types accepted by brower,encoding types supported by browser, accepted languages of browser,the type of browser etc..

Server can use these request headers to send proper response to the browser.

The following are some important request headers.

1. accept:media types accepted by browser (like html,pdf,images etc..)
2.accept-encoding: encoding types accepted by browser like gzip,dflate etc..
3.accept-language: languages types accepted by browser like un-en
4.user-agent: It represents the type of browser
5.cookie: used to send cookies for session management
etc...

utilities of request headers:

case-1:

By using accept-encoding request header,server can send compressed data instead of original data.It reduces download time and improves performance.

Eg:

```
String encoding=req.getHeader("accept-encoding");
if(encoding.equals("gzip"))
{
  send movie in gzip format
}
```

case-2:

By using user-agent request header,server can send browser specific customized response,so that we can increase hit rate of our application.

```
String userAgent=req.getHeader("user-agent");
if(userAgent.equals("mozilla"))
{
   provide mozilla compatible response
}
else if(userAgent.equals("chrome"))
{
```

provide chrome compatible response
}

case-3:
By using cookie request header,browser can send cookies to the server so that server can implement session management.

Note:
Browser will send request headers and server can use these request headers to provide proper response.

HttpServletRequest defines the following methods to retrieve header information at server side

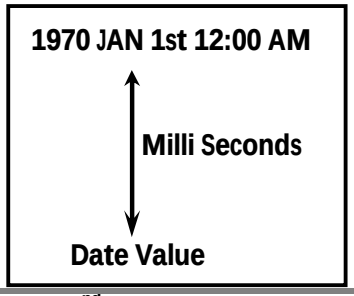
1. public String getHeader(String hname)
returns value associated with specified request header
If the specified request header is not already available then we will get null
If the header associated with multiple values then this method returns only first value.
argument is case insensitive
Eg:
String userAgent=req.getHeader("USER-AGENT");
String userAgent=req.getHeader("user-agent");

2. public Enumeration getHeaders(String hname):
returns all values associated with specified header.
If the specified header is not available then this method returns empty Enumeration object but not null.
argument is not case sensitive

3. public Enumeration getHeaderNames()
returns all header names associated with the request.

4. public int getIntHeader(String hname)
We can use this method if header associated with int value.
String length=req.getHeader("content-length");
int l = Integer.parseInt(length);
or
int l = req.getIntHeader("content-length");

5. public long getDateHeader(String hname)
We can use this method if header associated with Date value.
Returns Date value associated with the header as the number of milli seconds since jan 1st 1970.
If we pass these milliseconds as argument to Date constructor then we will get exact date.



Eg:

```
long l =req.getDateHeader("date");
Date d = new Date(l);
```

getHeader() getHeaders() getHeaderNames() getIntHeader() getDateHeader()

Demo Program to display all request headers information

RequestHeaderDemoServlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import java.util.*;
5)  import javax.servlet.annotation.*;
6)  @WebServlet("/test")
7)  public class RequestHeaderDemoServlet extends HttpServlet
8)  {
9)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
10)   {
11)      PrintWriter out = resp.getWriter();
12)      out.println("<h1>Request Headers Information</h1></hr>");
13)      out.println("<table border=2><tr><th>HeaderName</th><th>Header values</th></tr
    >");
14)      Enumeration e = req.getHeaderNames();
15)      while(e.hasMoreElements())
16)      {
17)         String hname = (String)e.nextElement();
18)         String hvalue=req.getHeader(hname);
19)         out.println("<tr><td>"+hname+"</td><td>"+hvalue+"</td></tr>");
20)      }
21)      out.println("</table></body></html>");
22)   }
23) }
```

```
advapps1E
   |-WEB-INF
      |-classes
            |-RequestHeaderDemoServlet.class
```

Retrieving Cookies from the request:

HttpServletRequest interface defines the following mehtods to retrieve cookies from the request.

```
Cookie[] c = req.getCookies();
```

Retrieving Client and Server information from the request:

We can get client and server information from the request by using the following methods of ServletRequest.

1.public String getRemoteHost()
Returns fully qualified name of the client which sends the request.

2. public String getRemoteAddr()
returns IP address of the client

3. public int getRemotePort()
returns the port number on which client is running

4. public String getServerName()
returns the server name to which request sent

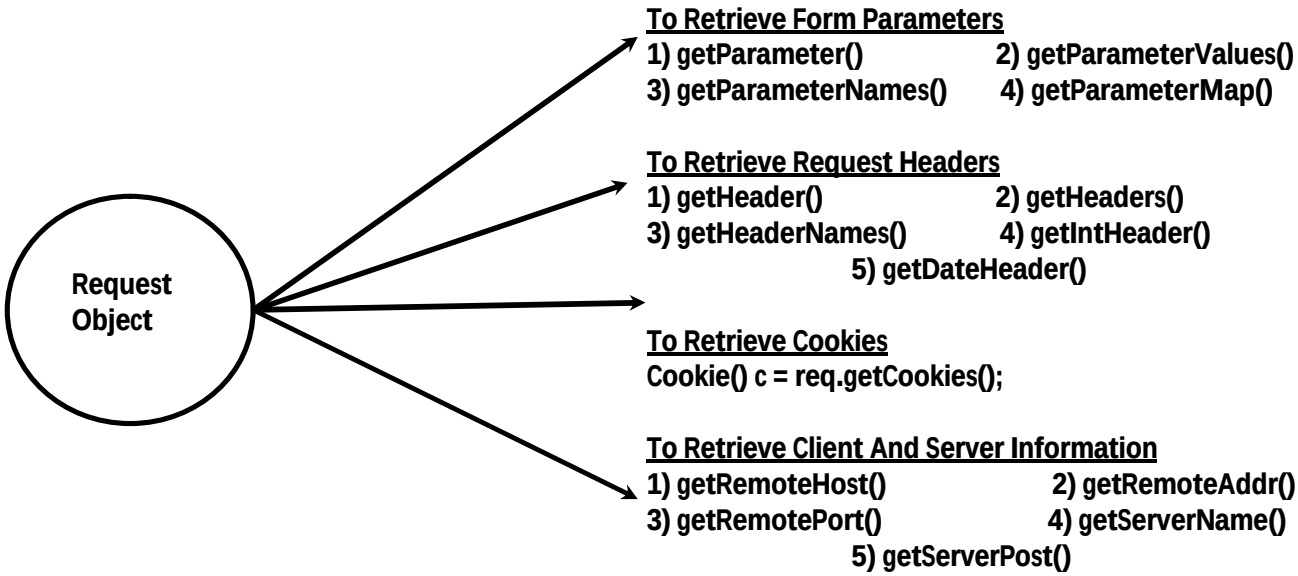
5. public int getServerPort()
returns port number at which server is running

Demo Program to print client and server information from the request:

ClientServerInfoServlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test")
6)  public class ClientServerInfoServlet extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
9)     {
10)       PrintWriter out = resp.getWriter();
11)       out.println("<h1>Client  IP Address:" +req.getRemoteAddr()+"</h1>");
12)       out.println("<h1>Client Host:" +req.getRemoteHost()+"</h1>");
13)       out.println("<h1>Client Port" +req.getRemotePort()+"</h1>");
14)       out.println("<h1>Server Name:" +req.getServerName()+"</h1>");
15)       out.println("<h1>Server Port:" +req.getServerPort()+"</h1>");
16)    }
17) }
```

```
advapps1E
   |-WEB-INF
      |-classes
            |-ClientServerInfoServlet.class
```



Playing Games with Response Object

Objective:

By using HttpServletResponse interface,write code
1. To set Response headers
2. To set Content type of Response
3. To acquire Text Stream for response
4. To acquire Binary Stream for response
5. To redirect request to another url
6. To add cookies to the response

1. To set Response headers:

Http Response headers provides configuration information of the server and information about the response like content-type,content-length etc
Browser will use these response headers to display response body properly to the end user.
HttpServletResponse interface defines the following methods to add headers to the response.

1. public void setHeader(String hname,String hvalue)
   If the specified header is already available then old value will be replaced with new value.

2. public void addHeader(String hname,String hvalue)
   If the specified header is already available then this new value also will be added to existing list of values. In this case replacement won't be happend.

HttpServletResponse interface defines the following extra methods to add int and date headers.

3. public void setIntHeader(String hname,int hvalue)

4. public void addIntHeader(String hname,int hvalue)

5. public void setDateHeader(String hname,long ms)

6. public void addDateHeader(String hname,long ms)

2.set content type of response:

ContentType header represents MIME type(multi purpose internet mail extension)of the response.

Common MIME Types are:

text/html==>Html text as response
application/pdf==>pdf file as response
application/msword==>msword file as response
application/vnd.ms-excel==>excel file as response
image/jpeg==>jpeg image file as response
video/mp4===>mp4 video file as response
etc...

We can set content Type of response in the following 2 ways

1. By ServletResponse:

ServletResponse interface defines the following method for this.

public void setContentType(String mimetype);

Eg:

```
resp.setContentType("text/html");
```

2. By HttpServletResponse:

HttpServletResponse interface defines the following method for this.

public void setHeader(String hname,String hvalue);

Eg:

```
resp.setHeader("content-type","text/html");
```

Demo Program to send HTML Response:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test")
6)  public class HtmlServlet extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
9)     {
10)       PrintWriter out = resp.getWriter();
11)       resp.setContentType("text/html");
```

```
12)       out.println("<h1><table border=1>");
13)       out.println("<tr><th>NAME</th><th>RANK</th></tr>");
14)       out.println("<tr><td>SUNNY</td><td>1</td></tr>");
15)       out.println("<tr><td>MALLIKA</td><td>2</td></tr>");
16)       out.println("<tr><td>VEENA</td><td>3</td></tr>");
17)       out.println("<tr><td>MALLIKA ARORA</td><td>4</td></tr></h1>");
18)    }
19) }
```

Demo Program to send Excel File as Response:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test")
6)  public class ExcelServlet extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
9)     {
10)       PrintWriter out = resp.getWriter();
11)       resp.setContentType("application/vnd.ms-excel");
12)       out.println("<h1><table border=1>");
13)       out.println("<tr><th>NAME</th><th>RANK</th></tr>");
14)       out.println("<tr><td>SUNNY</td><td>1</td></tr>");
15)       out.println("<tr><td>MALLIKA</td><td>2</td></tr>");
16)       out.println("<tr><td>VEENA</td><td>3</td></tr>");
17)       out.println("<tr><td>MALLIKA ARORA</td><td>4</td></tr></h1>");
18)    }
19) }
```

Demo Program to send Word File as Response:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test")
6)  public class WordServlet extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
9)     {
10)       PrintWriter out = resp.getWriter();
11)       resp.setContentType("application/msword");
12)       out.println("<h1><table border=1>");
13)       out.println("<tr><th>NAME</th><th>RANK</th></tr>");
14)       out.println("<tr><td>SUNNY</td><td>1</td></tr>");
```

```
15)       out.println("<tr><td>MALLIKA</td><td>2</td></tr>");
16)       out.println("<tr><td>VEENA</td><td>3</td></tr>");
17)       out.println("<tr><td>MALLIKA ARORA</td><td>4</td></tr></h1>");
18)    }
19) }
```

3. To acquire text stream for the response:

We can send text data as the response by using PrintWriter object.
We can get PrintWriter object by using getWriter() method of ServletResponse interface.

Eg:

```
PrintWriter out=resp.getWriter();
out.println("Hello this is text response from the servlet");
```

4. To acquire binary stream for the response:

We can send binary information(like video files,audio files,images etc..)as response by using ServletOutputStream object.
We can create ServletOutputStream object by using getOutputStream() method of ServletResponse interface.

Eg: `ServletOutputStream os=resp.getOutputStream();`

BinaryStreamDemo1Servlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  public class BinaryStreamDemo1Servlet extends HttpServlet
5)  {
6)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
7)     {
8)        resp.setContentType("image/JPEG");
9)        ServletOutputStream os = resp.getOutputStream();
10)       String path=getServletContext().getRealPath("sunny.jpg");
11)       File f = new File(path);
12)       FileInputStream fis = new FileInputStream(f);
13)       byte[] b = new byte[(int)f.length()];
14)       fis.read(b);//reading the image and placing into byte array
15)                   os.write(b);//write byte array to the respose
16)       os.flush();
17)       os.close();
18)    }
19) }
```

```
advapps1G
   |- sunny.jpg
   |-WEB-INF
      |-classes
         |-BinaryStreamDemoServlet.class
```

Demo Program to send Video file as Response:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test")
6)  public class BinaryStreamDemo1Servlet extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
9)     {
10)       resp.setContentType("video/mp4");
11)       ServletOutputStream os = resp.getOutputStream();
12)       String path=getServletContext().getRealPath("dhoni.mp4");
13)       File f = new File(path);
14)       FileInputStream fis = new FileInputStream(f);
15)       byte[] b = new byte[(int)f.length()];
16)       fis.read(b);//reading the image and placing into byte array
17)                   os.write(b);//write byte array to the respose
18)       os.flush();
19)       os.close();
20)    }
21) }
```

```
advapps1GA
   |- dhoni.mp4
   |-WEB-INF
      |-classes
            |-BinaryStreamDemoServlet.class
```

Note:

By using PrintWriter we can send text data as response but by using ServletOutputStream we can send both text data and binary data as response.

Note:

Within the servlet we can get either PrintWriter or ServletOutputStream but not both simultaneously.

If we are trying to get both then we will get RuntimeException saying IllegalStateException.

Eg:

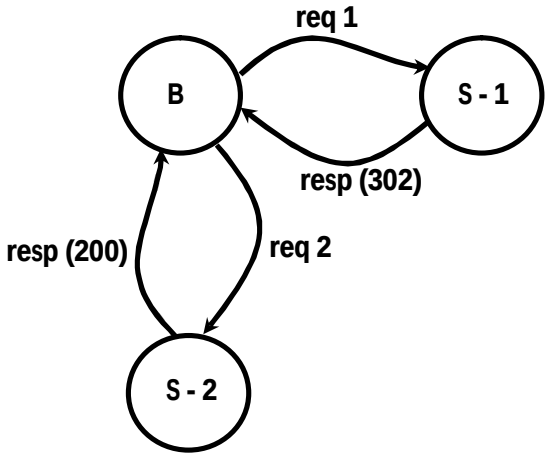
```
public void doGet(..)...
{
   PrintWriter out=resp.getWriter();
   ServletOutputStream os = resp.getOutputStream();
}
RE: java.lang.IllegalStateException: getWriter() has already been called for this response
```

Redirecting Http Request to another url:

Sometimes we have to redirect Http request to another url.We can implement this by using sendRedirect() method HttpServletResponse interface.

```
public void sendRedirect(String targetPath) throws IOE
```

Process of Redirection:



1. Browser sends a request to servlet-1
2. If servlet-1 is not responsible to provide required response ,then it will update browser to contact servlet-2.This can be informed by setting status code 302 and location header with servlet-2.
3.By seeing 302 status code, browser creates a new request and sends to servlet-2.
4. Servlet-2 provides required response(200) to the browser and browser inturn provides that response to the end user.

Demo Program for Redirection:

FirstServlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test1")
6)  public class FirstServlet extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
9)     {
10)       //resp.sendRedirect("/advapps1H/test2");
11)       resp.setStatus(302);
12)       resp.setHeader("Location","/advapps1H/test2");
13)    }
14) }
```

SecondServlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test2")
6)  public class SecondServlet extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
9)     {
10)       PrintWriter out =resp.getWriter();
11)       out.println("<h1>Hi This is Second Servlet</h1>");
12)    }
13) }
```

```
advapps1H
   |-WEB-INF
      |-classes
            |-FirstServlet.class
            |-SecondServlet.class
```

In the above example whenever we are sending the request to FirstServlet,SecondServlet will provide required response b'z of redirection.

*****Note:**
After committing the response we are not allowed to perform redirection,otherwise we will get RuntimeException saying IllegalStateException.(But this feature is not implemented by Tomcat)

Note: sendRedirection is happening at client side(browser side)

Eg of redirection:
Whenever we are sending the request to google.com, our request will be redirected to google.co.in

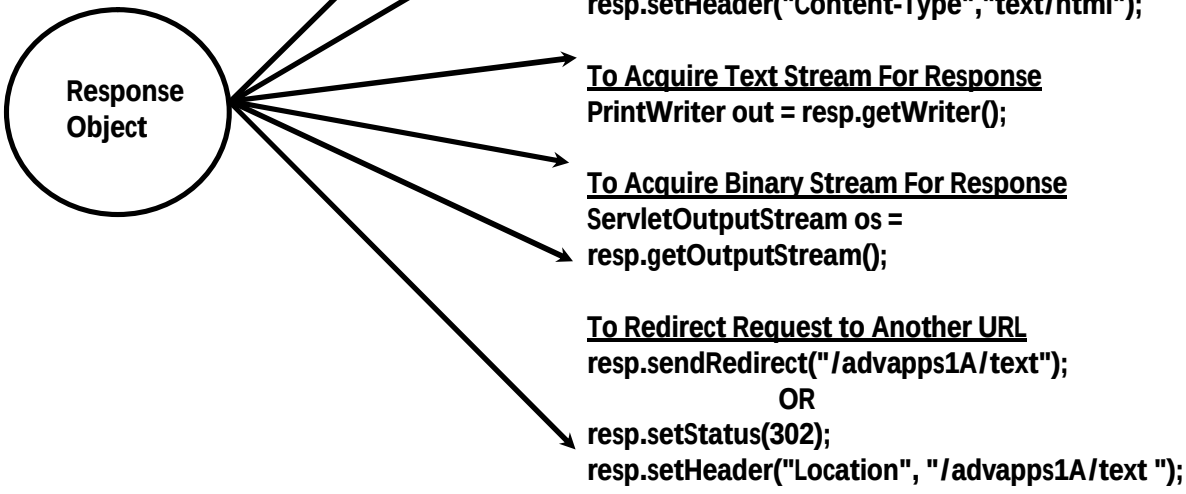
Adding Cookies to Response:
Cookie is key-value pair. We can use Cookies in session management.

We can create Cookie object as follows...

```
Cookie c = new Cookie("durga","java");
```

We can add Cookie to the response by using addCookie() method of HttpServletResponse.

```
resp.addCookie(c);
```


Response Object
To Set Response Headers
1) setHeader() 2) addHeader()
3) setIntHeader() 4) addIntHeader()
5) setDateHeader() 6) addDateHeader()
To Set Content-Type of Response
resp.setContentType("text/html");
OR
resp.setHeader("Content-Type","text/html");
To Acquire Text Stream For Response
PrintWriter out = resp.getWriter();
To Acquire Binary Stream For Response
ServletOutputStream os =
resp.getOutputStream();
To Redirect Request to Another URL
resp.sendRedirect("/advapps1A/text");
OR
resp.setStatus(302);
resp.setHeader("Location", "/advapps1A/text ");

UNIT-2: Directory Structure and Deployment of Web Application

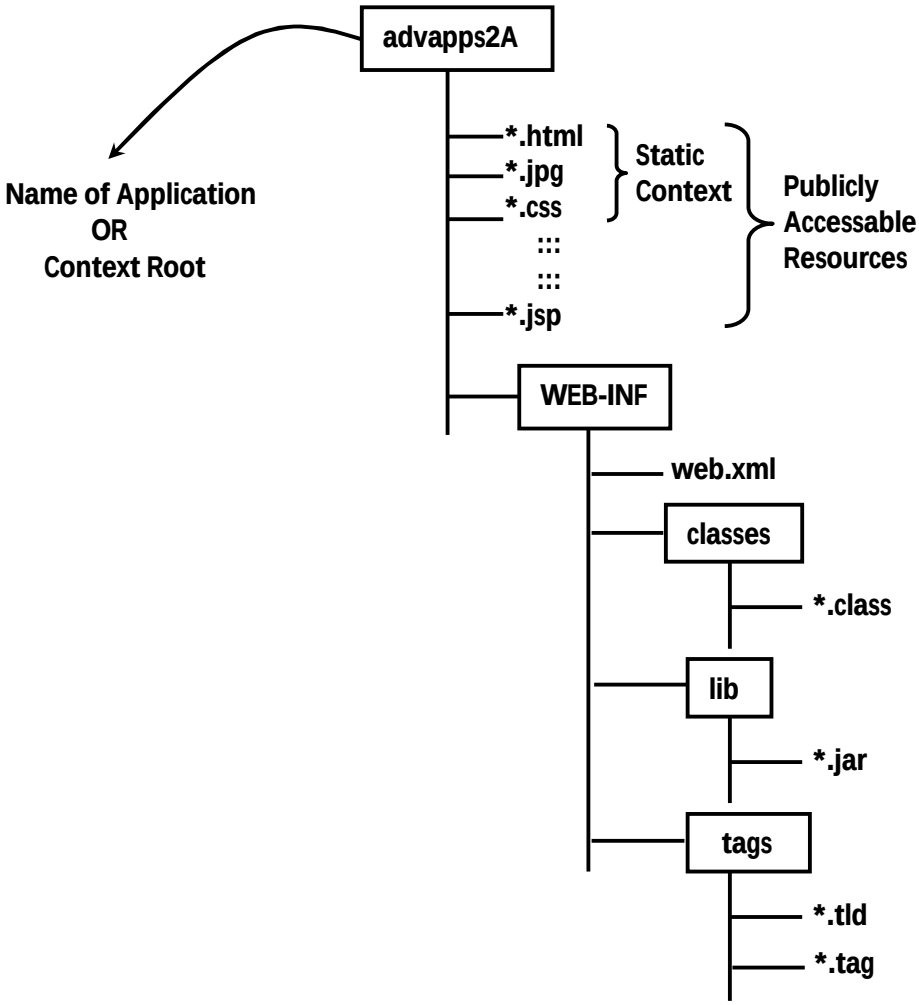
1. Directory Structure of web application
2. web.xml
3. war file

1. Directory Structure of web application:

Objective:

Construct File and Directory Structure of web application that may contain

1. Static Content
2. JSP Pages
3. Servlet classes
4. Deployment Descriptor
5. Tag Libraries
6. Jar Files
7. Java Class Files



Servlet Specification defines a standard structure for web applications so that every web server can provide support for that structure irrespective of vendor.

Total static information we have to place within the context root directly. All resources present in context root are publicly accessable resources. ie any person can access these resouces directly by its name.Hence we cannot place secured resouces within the context root directly.

http://localhost:7777/advapps1C/login.html

For every web application we have to maintain WEB-INF Folder. All the secured resources we have to place in this folder and these are not publicly accessable resources. If any person trying to access these resouces directly by its name then we will get 404 status code.

http://localhost:7777/advapps1B/WEB-INF/web.xml ==>404

For every web application, we have to maintain one special xml file named with web.xml. It is also known as Deployment Descriptor. Web container will use this web.xml to get our servlets information.Hence this web.xml acts as GUIDE to web container.
We have to place this web.xml in WEB-INF folder directly.

Within the WEB-INF, we have to create classes folder to place all .class files.(Both Servlet classes and normal java classes)

Within the WEB-INF, we have to maintain lib folder to place all required .jar files(like ojdbc6.jar,log4j.jar etc)

Note:
if we place our class files in classes folder and lib folder, we are not required to set classpath explicitly, b'z web container always searches in these locations by default.

Note:
For the required .class file web container will search first in classes folder and if it is not available then only it will search in lib folder.i.e classes present in classes folder will get more priority than classes present in lib folder.

All Tag libraries we have to place inside WEB-INF either directly or indirectly.

Resourse	Location
1) Static Context	1) Within the Context Root directly
2) JSP Pages	2) Within the Context Root directly. If JSP is secured then we have to place inside WEB-INF
3) Servlet Classes	3) Inside Classes Folder
4) Deployment Descriptor (web.xml)	4) Inside WEB-INF directly
5) Tag Libraries	5) Within WEB-INF either directly OR indirectly
6) JAR Files	6) Inside lib Folder
7) Java Class Files	7) Inside Classes Folder

Q. In how many places we can place our servlet in web application?

There are 3 places
1. Inside classes folder(.class)
2. Inside lib folder(.jar)
3. Anywhere we can place but we have to set classpath explicitly.

Deployment Descriptor: (web.xml)

Objective:

Describe the purpose and syntax of the following deployment descriptor elements?

1.<error-page>
2.<init-param>
3.<mime-mapping>
4.<servlet>
5.<servlet-mapping>
6.<servlet-class>
7.<servlet-name>
8.<welcome-file>

Deployment Descriptor(DD) is an XML File named with web.xml,should be placed in WEB-INF directly.

Web container uses this web.xml to get web application deployment information. i.e web.xml acts as GUIDE to web container.

web.xml provides declarative mechanism for customizing our web application without touching the source code(with the help of initialization parameters).

For every web application we have to maintain exactly one web.xml and should be placed in WEB-INF folder directly.

Anatomy of web.xml:

```
1)  <web-app>
2)    <description>
3)    <display-name>
4)    <servlet>
5)    <servlet-mapping>
6)    <welcome-file-list>
7)    <error-page>
8)    <mime-mapping>
9)    <context-param>
10)   <security-constraint>
11)   <jsp-config>
12)   <session-config>
13)   <filter>
14)   <filter-mapping>
```

Around 27 Top Level Tags

```
15)  <listener>
16)  ...
17) </web-app>
```

until servlet 2.3 version,web.xml is validated by using DTDs where order is important.Hence until Servlet 2.3V the order of these top level tags is important.

But from Servlet 2.4V onwards web.xml is validated by using Schemas where order is not important.Hence from servlet 2.4 Version onwards we can take these top level tags in any order.

Among these 27 top level tags , no tag is mandatory.Hence the following are valid web.xml files

<web-app> </web-app>	OR	<web-app/>

1.<servlet>:

We can use <servlet> tag to declare servlets in web.xml.
<servlet> tag contains the following 9 child tags...

```
1)  <servlet>
2)    <description>
3)    <display-name>
4)    <icon>
5)    <servlet-name>
6)    <servlet-class> or <jsp-file>      } 9 Child Tags
7)    <init-param>
8)    <load-on-startup>
9)    <run-as>
10)   <security-role-ref>
11) </servlet>
```

The order of these 9 tags is important and we should not change.

Among these 9 tags the following 2 tags are mandatory.

<servlet-name>
<servlet-class> or <jsp-file>

<servlet-name>:

According to Servlet specification there are 3 names are possible for every servlet.

```
1)  <web-app>
2)    <servlet>
3)       <servlet-name>DemoServlet</servlet-name>
4)       <servlet-class>FirstServlet</servlet-class>
5)    </servlet>
6)
7)    <servlet-mapping>
8)       <servlet-name>DemoServlet</servlet-name>
9)     <url-pattern>/test</url-pattern>
10)   </servlet-mapping>
11) </web-app>
```

Deployer's Name OR Logical

Original Name OR Developer's Name

End User's Name

1. Original Name OR Developer's Name specified by <servlet-class>

2. Deployer's Name OR Logical Name specified by <servlet-name> tag
3. End User's Name speficied by <url-pattern>

The main advantages of Logical Name are:
1. We can achieve security because we are not highlighting our internal naming conventions to the end user.

2.Without effecting end user we can change our internal naming conventions.

3.We can map same servlet with different url-patterns

Note:

Within web.xml Logical Name is always unique.

Within the servlet we can get Logical Name by using getServletName() method of ServletConfig interface.

```
1)  public class FirstSevlet extends HttpServlet
2)  {
3)     public void doGet(..)...
4)     {
5)        PrintWriter out = resp.getWriter();
6)        String name=getServletName();
7)
8)        out.println("<h1>Logical Name:"+name+"</h1>");
9)     }
10) }
```

Note:

All methods of ServletConfig interface are by default available in our servlet.Hence within our servlet we can call these methods directly.

<servlet-class>:

By using this tag,we can specify the original name of servlet(Developer's name)

Web container will create object for this class only.Hence compulsary it should be concrete class but not abstract class or interface.

Configuring JSP in web.xml:

Usually we can place JSPs within the context root directly. In this case end user can access directly by its name.

But if the JSP is secured then it is not recommended to place in the context root directly. We have to place such type of jsps inside WEB-INF and we have to provide access through url-pattern. We can configure JSP by using <jsp-file> tag in web.xml

Demo Program:

date.jsp:

```
1)  <h1>Now Server Time is :
2)     <%=new java.util.Date()%>
3)  </h1>
```

web.xml:

```
1)  <web-app>
2)     <servlet>
3)       <servlet-name>FirstJSP</servlet-name>
4)       <jsp-file>/WEB-INF/date.jsp</jsp-file>
5)       <init-param>
6)       </init-param>
7)     </servlet>
8)     <servlet-mapping>
9)       <servlet-name>FirstJSP</servlet-name>
10)     <url-pattern>/test</url-pattern>
11)   </servlet-mapping>
12) </web-app>
```

```
advapps2B
   |--WEB-INF
         |--date.jsp
         |--web.xml
```

http://localhost:7777/advapps2B/test==>valid
http://localhost:7777/advapps2B/WEB-INF/date.jsp==>404

Servlet Initialization Parameters(<init-param>):

If the value of the variable will change frequently then those values are not recommended to hard-code within the servlet class.

The problem in this approach is, If there is any change in the value, to reflect that change,we have to recompile,rebuild and redeploy application, and sometimes even server restart also required,which creates big business impact to the client.

Such type of variables we have to configure in web.xml by using <init-param> tag.

The advantage in this approach is if there is any change in the value, to reflect that change just redeployment is enough,which won't create big business impact to the client.

We can declare servlet initialization parameters in web.xml as follows...

```
1)  <web-app>
2)
3)      <servlet>
4)          <servlet-name>FirstServlet</servlet-name>
5)          <servlet-class>FirstServlet</servlet-class>
6)          <init-param>
7)              <param-name>user</param-name>
8)              <param-value>scott</param-value>
9)          </init-param>
10)         <init-param>
11)             <param-name>pwd</param-name>
12)             <param-value>tiger</param-value>
13)         </init-param>
14)     </servlet>
15)
16) </web-app>
```

We can declare any number of init parameters but for each parameter one <init-param> tag.
Within the servlet we can access servlet initialization parameters by using ServletConfig object.
ServletConfig interface defines the following methods to access these parameters inside servlet.

1. public String getInitParameter(String pname)
   Returns the value associated with specified parameter.
   Returns null if the specified paramter is not avaialble.

2. public Enumeration getInitParameterNames()
   Returns all initialization parameter names.
   If the specified servlet does not contain any initialization parameters then this method returns empty enumeration object but not null.

GenericServlet implements ServletConfig interface and hence GenericServlet provides implementation for the above 2 methods.Within our servlet we can call these methods directly.

Demo Program to print all Servlet Initialization Parameters:

web.xml:

```
<web-app>
 <servlet>
   <servlet-name>DemoServlet</servlet-name>
   <servlet-class>InitializeParameterDemoServlet</servlet-class>
   <init-param>
     <param-name>PhoneNumber</param-name>
     <param-value>9848012345</param-value>
   </init-param>
   <init-param>
     <param-name>MailId</param-name>
     <param-value>durgaocjp@gmail.com</param-value>
   </init-param>
   <init-param>
     <param-name>UserName</param-name>
     <param-value>scott123</param-value>
   </init-param>
</servlet>
<servlet-mapping>
 <servlet-name>DemoServlet</servlet-name>
 <url-pattern>/test</url-pattern>
</servlet-mapping>
 </web-app>
```

InitializeParameterDemoServlet.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class InitializeParameterDemoServlet extends HttpServlet
{
   public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
eption,IOException
   {
     PrintWriter out = resp.getWriter();
     out.println("<center><h1>Intialization Parameters</h1></center><hr>");
     Enumeration e = getInitParameterNames();
     out.println("<table border=2><tr><th>Parameter Name</th><th>Parameter  Value</t
h> </tr>");
     while (e.hasMoreElements())
     {
       String pname = (String)e.nextElement();
       String pvalue = getInitParameter(pname);

       out.println("<tr><td>"+pname+"</td><td>"+pvalue+"</td></tr>");
```

```
19.       }
20.       out.println("</table>");
21.       out.println("</body></html>");
22.     }
23.  }
```

```
advapps2C
   |-WEB-INF
      |-web.xml
      |-classes
            |-InitializeParameterDemoServlet.class
```

Demo Program to print total number of employees from the database by Servlet Initialization Parameters:

web.xml:

```
1.  <web-app>
2.   <servlet>
3.      <servlet-name>FirstServlet</servlet-name>
4.      <servlet-class>FirstServlet</servlet-class>
5.      <init-param>
6.           <param-name>user</param-name>
7.           <param-value>scott</param-value>
8.      </init-param>
9.      <init-param>
10.          <param-name>pwd</param-name>
11.          <param-value>tiger</param-value>
12.     </init-param>
13.  </servlet>
14. <servlet-mapping>
15.   <servlet-name>FirstServlet</servlet-name>
16.   <url-pattern>/test</url-pattern>
17. </servlet-mapping>
18.  </web-app>
```

FirstServlet.java:

```
1.  import javax.servlet.*;
2.  import javax.servlet.http.*;
3.  import java.io.*;
4.  import java.sql.*;
5.  public class FirstServlet extends HttpServlet
6.  {
7.     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
8.     {
9.        String user=getInitParameter("user");
```

```
10.       String pwd=getInitParameter("pwd");
11.       PrintWriter out = resp.getWriter();
12.       try{
13.       Class.forName("oracle.jdbc.OracleDriver");
14.       Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
      ",user,pwd);
15.       Statement st =con.createStatement();
16.       ResultSet rs =st.executeQuery("select count(*) from employees");
17.       while(rs.next())
18.       {
19.          int count=rs.getInt(1);
20.          out.println("<h1>The number of employees:"+count+"</h1>");
21.       }
22.       }
23.       catch(Exception e)
24.       {
25.          e.printStackTrace();
26.       }
27.    }
28. }
```

```
advapps2CB
   |-WEB-INF
      |-web.xml
      |-lib
            |-ojdbc6.jar
      |-classes
            |-FirstServlet.class
```

Note:

Servlet initialization parameters are key-value pairs where both key and value are String type.

From the servlet we can access these parameters but we cannot modify. i.e we have only getter methods but not setter methods. Hence Servlet initialization parameters are considered as deployment time constants.

ServletConfig (I):

For every servlet web container creates one ServletConfig object to hold its configuration information.

By using ServletConfig object, servlet can get its configuration information.

ServletConfig defines the following 4 methods

1. public String getServletName()
2. public String getInitParameter()
3. public Enumeration getInitParamterNames()
4. public ServletContext getServletContext()

<load-on-startup>:

usually servlet class loading,instanitation and execution of init() method will takes place at the time of first request. It increases processing time of first request when compared with remaining requests.

To overcome this problem, we should go for <load-on-startup>

If we configured <load-on-startup> then these steps will be performed at the time of server start up or at application deployment.

```
1) <web-app>
2)    <servlet>
3)       ....
4)       <load-on-startup>10</load-on-startup>
5)    </servlet>
6)    ..
7) </web-app>
```

The main advantage of <load-on-startup> is all requests will be processed with uniform response time.

The main disadvantage of <load-on-startup> is, creating servlet object at the beginning may effect performance and causes memory problems.

The servlet whose <load-on-startup> value is less, that servlet will be loaded first.

If 2 servlets having same <load-on-startup> value or if the <load-on-startup> value is negative then we cannot expect order of loading. It is web server vendor dependent.

Demo Program to demonstrate <load-on-startup>:

FirstServlet.java:

```
1.  import javax.servlet.*;
2.  import java.io.*;
3.  public class FirstSevlet extends GenericServlet
4.  {
5.     static
6.     {
7.        System.out.println("FirstServlet Loading..");
8.     }
9.     public FirstSevlet()
10.    {
11.       System.out.println("FirstServlet Instantiation..");
12.    }
13.    public void init(ServletConfig config) throws ServletException
14.    {
15.       System.out.println("FirstServlet init method execution..");
```

```
16.    }
17.    public void service(ServletRequest req, ServletResponse resp) throws ServletException,
       IOException
18.    {
19.       PrintWriter out = resp.getWriter();
20.       out.println("<h1>FirstServlet:Writing servlet by extending GS is very easy</h1>");
21.       System.out.println("service method called..");
22.    }
23. }
```

SecondServlet.java:

```
1.  import javax.servlet.*;
2.  import java.io.*;
3.  public class SecondServlet extends GenericServlet
4.  {
5.     static
6.     {
7.        System.out.println("SecondServlet Loading..");
8.     }
9.     public SecondServlet()
10.    {
11.       System.out.println("SecondServlet Instantiation..");
12.    }
13.    public void init(ServletConfig config) throws ServletException
14.    {
15.       System.out.println("SecondServlet init method execution..");
16.    }
17.    public void service(ServletRequest req, ServletResponse resp) throws ServletException,
       IOException
18.    {
19.       PrintWriter out = resp.getWriter();
20.       out.println("<h1>SecondServlet:Writing servlet by extending GS is very easy</h1>");
21.       System.out.println("service method called..");
22.    }
23. }
```

web.xml:

```
1.  <web-app>
2.  <servlet>
3.    <servlet-name>FirstSevlet</servlet-name>
4.    <servlet-class>FirstSevlet</servlet-class>
5.    <load-on-startup>20</load-on-startup>
6.  </servlet>
7.  <servlet>
8.    <servlet-name>SecondServlet</servlet-name>
9.    <servlet-class>SecondServlet</servlet-class>
10.   <load-on-startup>10</load-on-startup>
```

```
</servlet>
<servlet-mapping>
  <servlet-name>FirstSevlet</servlet-name>
  <url-pattern>/test</url-pattern>
</servlet-mapping>
</web-app>
```

advapps2D
|-WEB-INF
|-web.xml
|-classes
|-FirstServlet.class
|-SecondServlet.class

<servlet-mapping>:
By using this tag, we can map a servlet with <url-pattern>.

Up to Servlet 2.4 version within <servlet-mapping>, a single servlet can map to exactly one <url-pattern>

But from Servlet 2.5V onwards we can map servlet with multiple url patterns. i.e we can take mulitple <url-pattern> tags inside <servlet-mapping>.

```
<servlet-mapping>
    <servlet-name>DemoServlet</servlet-name>
    <url-pattern>/test</url-pattern>
    <url-pattern>/hello</url-pattern>
    <url-pattern>/demo</url-pattern>
</servlet-mapping>
```

It is invalid in Servlet 2.4V but valid in Servlet 2.5V.

According to Servlet specification there are 4 types of url-patterns are possible.

1. Exact match url-pattern
eg: /test

2. Longest Path Prefix url-pattern or directory match url-pattern

eg: /test/test/*

3. url-pattern by extension
eg: *.do

4. Default url-pattern
eg: /

Q.Which of the following are valid url-patterns ?

1. /test ✔
2. /test/*/test ✕
3. /test/test/* ✔
4. *.task ✔
5. / ✔
6. /test/test/*.do ✕

***Web container always gives the precedence in the following order

1. Exact Match UP
2. Longest Path Prefix UP
3. UP by extension
4. Default UP

Demo Program to demonistrate different types of url-patterns and priority order:

FirstServlet.java:

```
1.  import javax.servlet.*;
2.  import javax.servlet.http.*;
3.  import java.io.*;
4.  public class FirstServlet extends HttpServlet
5.  {
6.     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
7.     {
8.        PrintWriter out = resp.getWriter();
9.        out.println("<h1>Hi ...This is First Servlet</h1>");
10.    }
11. }
```

SecondServlet.java:

```
1.  import javax.servlet.*;
2.  import javax.servlet.http.*;
3.  import java.io.*;
4.  public class SecondServlet extends HttpServlet
5.  {
6.     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
7.     {
8.        PrintWriter out = resp.getWriter();
9.        out.println("<h1>Hi ...This is Second Servlet</h1>");
10.    }
11. }
```

ThirdServlet.java:

```
1.  import javax.servlet.*;
2.  import javax.servlet.http.*;
3.  import java.io.*;
4.  public class ThirdServlet extends HttpServlet
5.  {
6.     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption, IOException
7.     {
8.        PrintWriter out = resp.getWriter();
9.        out.println("<h1>Hi ...This is Third Servlet</h1>");
10.    }
11. }
```

DefaultServlet.java:

```
1.  import javax.servlet.*;
2.  import javax.servlet.http.*;
3.  import java.io.*;
4.  public class DefaultServlet extends HttpServlet
5.  {
6.     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
7.     {
8.        PrintWriter out = resp.getWriter();
9.        out.println("<h1>Hi ...This is Default Servlet</h1>");
10.    }
11. }
```

web.xml:

```
1.  <web-app>
2.  <servlet>
3.    <servlet-name>FirstServlet</servlet-name>
4.    <servlet-class>FirstServlet</servlet-class>
5.  </servlet>
6.  <servlet>
7.    <servlet-name>SecondServlet</servlet-name>
8.    <servlet-class>SecondServlet</servlet-class>
9.  </servlet>
10. <servlet>
11.   <servlet-name>ThirdServlet</servlet-name>
12.   <servlet-class>ThirdServlet</servlet-class>
13. </servlet>
14. <servlet>
15.   <servlet-name>DefaultServlet</servlet-name>
16.   <servlet-class>DefaultServlet</servlet-class>
17. </servlet>
```

```
18. <servlet-mapping>
19.   <servlet-name>FirstServlet</servlet-name>
20.   <url-pattern>/test</url-pattern>
21. </servlet-mapping>
22. <servlet-mapping>
23.   <servlet-name>SecondServlet</servlet-name>
24.   <url-pattern>/test/test/*</url-pattern>
25. </servlet-mapping>
26. <servlet-mapping>
27.   <servlet-name>ThirdServlet</servlet-name>
28.   <url-pattern>*.do</url-pattern>
29. </servlet-mapping>
30. <servlet-mapping>
31.   <servlet-name>DefaultServlet</servlet-name>
32.   <url-pattern>/</url-pattern>
33. </servlet-mapping>
34. </web-app>
```

```
advapps2E
   |-WEB-INF
      |-web.xml
      |-classes
            |-FirstServlet.class
            |-SecondServlet.class
            |-ThirdServlet.class
            |-DefaultServlet.class
```

FS==>/test
SS==>/test/test/*
TS==>*.do
DS===>/

http://localhost:7777/advapps2E/test ==>FS
http://localhost:7777/advapps2E/test/test/durga ==>SS
http://localhost:7777/advapps2E/test/test/durga.do ==>SS

http://localhost:7777/advapps2E/test/durga.do ==>TS

http://localhost:7777/advapps2E/test/anushka ==>DS

Q. How to configure Default Servlet in web.xml and when it will get chance?

We can configure Default Servlet with url-pattern "/".
If no other servlet matched then only default servlet will get chance.

Getting Extra Information from the url:

ServletRequest interface defines the following methods for this

1. getRequestURI()
2. getContextPath()
3. getServletPath()
4. getPathInfo()
5. getQueryString()

Demo Program to print extra information from the url:

FirstServlet.java:

```
1.  import javax.servlet.*;
2.  import javax.servlet.http.*;
3.  import java.io.*;
4.  public class FirstServlet extends HttpServlet
5.  {
6.     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
7.     {
8.        PrintWriter out = resp.getWriter();
9.        out.println("<h1>Request URI:" +req.getRequestURI()+"</h1>");
10.       out.println("<h1>Context Path:" +req.getContextPath()+"</h1>");
11.       out.println("<h1>Servlet Path:" +req.getServletPath()+"</h1>");
12.       out.println("<h1>Path Info:" +req.getPathInfo()+"</h1>");
13.       out.println("<h1>Query String:" +req.getQueryString()+"</h1>");
14.    }
15. }
```

web.xml

```
1.  <web-app>
2.  <servlet>
3.    <servlet-name>FirstServlet</servlet-name>
4.    <servlet-class>FirstServlet</servlet-class>
5.  </servlet>
6.  <servlet-mapping>
7.    <servlet-name>FirstServlet</servlet-name>
8.    <url-pattern>/test/test/*</url-pattern>
9.  </servlet-mapping>
10. </web-app>
```

```
advapps2F
   |-WEB-INF
      |-web.xml
      |-classes
            |-FirstServlet.class
```

FS==>/test/test/*

Eg1:
http://localhost:7777/advapps2F/test/test/durga/software?user=durga&pwd=anushka

Request URI:/advapps2F/test/test/durga/software
Context Path:/advapps2F
Servlet Path:/test/test
Path Info:/durga/software
Query String:user=durga&pwd=anushka

http://localhost:7777/advapps2F/test/test/durga/software

Request URI:/advapps2F/test/test/durga/software
Context Path:/advapps2F
Servlet Path:/test/test
Path Info:/durga/software
Query String:null

http://localhost:7777/advapps2F/test/test

Request URI:/advapps2F/test/test/durga/software
Context Path:/advapps2F
Servlet Path:/test/test
Path Info: null
Query String:null

Configuring welcome files for web application:

It is highly recommended to configure welcome files for our web application. It increases easyness to use our web application for the end user.

We can configure welcome files in the web.xml as follows...

```
1)  <web-app>
2)    <welcome-file-list>
3)        <welcome-file>home.jsp</welcome-file>
4)        <welcome-file>login.jsp</welcome-file>
5)        <welcome-file>index.jsp</welcome-file>
6)    </welcome-file-list>
7)  </web-app>
```

<welcome-file-list> is the direct child tag of <web-app> and hence we can take anywhere.
We can configure any number of welcome files but the order is important. Web container always consider from top to bottom.
In any web application index.html acts as default welcome file.If index.html is not available then index.jsp acts as default welcome file.

If we configured welcome files explicitly then default welcome files concept is not applicable.
welcome files concept is applicable for sub folders also.

Demo Program to demonstrate default welcome files:

index.jsp:

```
<h1>Welcome to Durgajobs information<br/><hr/>
<a href="/advapps2GW/test1">Hyderabad Jobs Info</a><br/>
<a href="/advapps2GW/test2">Bangalore Jobs Info</a><br/>
<a href="/advapps2GW/test3">Pune Jobs Info</a></h1>
```

HydJobsServlet.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class HydJobsServlet extends HttpServlet
{
   public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletException,IOException
   {
      PrintWriter out = resp.getWriter();
      out.println("<h1>Hyderabad Jobs Info</h1>");
   }
}
```

BangaloreJobsServlet.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class BangaloreJobsServlet extends HttpServlet
{
   public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletException,IOException
   {
      PrintWriter out = resp.getWriter();
      out.println("<h1>Bangalore Jobs Info</h1>");
   }
}
```

PuneJobsServlet.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class PuneJobsServlet extends HttpServlet
{
   public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletException,IOException
   {
```

```
        PrintWriter out = resp.getWriter();
        out.println("<h1>Pune Jobs Info</h1>");
    }
}
```

web.xml:

```
<web-app>
<servlet>
  <servlet-name>FirstServlet</servlet-name>
  <servlet-class>HydJobsServlet</servlet-class>
</servlet>
<servlet>
  <servlet-name>SecondServlet</servlet-name>
  <servlet-class>BangaloreJobsServlet</servlet-class>
</servlet>
<servlet>
  <servlet-name>ThirdServlet</servlet-name>
  <servlet-class>PuneJobsServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>FirstServlet</servlet-name>
  <url-pattern>/test1</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>SecondServlet</servlet-name>
  <url-pattern>/test2</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>ThirdServlet</servlet-name>
  <url-pattern>/test3</url-pattern>
</servlet-mapping>
</web-app>
```

```
advapps2GW
  |-WEB-INF
    |-web.xml
    |-classes
        |-HydJobsServlet.class
        |-BangaloreJobsServlet.class
        |-PuneJobsServlet.class
```

http://localhost:7777/advapps2GW

Demo Program to demonistrate customized welcome files:

```
advapps2G
  |-home.jsp
  |-durga
```

```
    |-login.jsp
    |-durga1
       |-home.jsp
       |-login.jsp
  |-durga2
     |-index.jsp
  |-durga3
     |-durga2.jsp
     |-form.html
  |-WEB-INF
     |-web.xml
```

web.xml:

```
1.  <web-app>
2.    <welcome-file-list>
3.       <welcome-file>home.jsp</welcome-file>
4.       <welcome-file>login.jsp</welcome-file>
5.       <welcome-file>index.jsp</welcome-file>
6.    </welcome-file-list>
7.  </web-app>
```

http://localhost:7777/advapps2G ==>home.jsp

http://localhost:7777/advapps2G/durga==>login.jsp

http://localhost:7777/advapps2G/durga/durga1==>home.jsp
http://localhost:7777/advapps2G/durga2==>index.jsp

http://localhost:7777/advapps2G/durga3==>404 Status code

Note:
According to Servlet Specification the value of <welcome-file> tag should be name of the jsp but not location. Hence it should not starts with "/"

Eg:
<welcome-file>home.jsp</welcome-file>==>valid
<welcome-file>/home.jsp</welcome-file>===>invalid

Configuring Error Pages in web.xml:

It is not recommended to send error information directly to the end user. We have to convert that java specific error information into end user understandable form.

We can achieve this by configuring error pages in web.xml.

We can configure error page either based on exception type or based on error code.

Configuring error page based on exception type:

```
1)  <web-app>
2)     <error-page>
3)       <exception-type>java.lang.ArithmeticException</exception-type>
4)       <location>/error.jsp</location>
5)     </error-page>
6)  </web-app>
```

Configuring error page based on error code:

```
1)  <web-app>
2)     <error-page>
3)       <error-code>404</error-code>
4)       <location>/test1</location>
5)     </error-page>
6)  </web-app>
```

Note:

1. <error-page> is the direct child tag of <web-app> and hence we can take anywhere within <web-app>.
2. error page can be either servlet or jsp.

Demo Program for configuring error pages in web.xml:

FirstServlet.java:

```
1)  public class FirstSevlet extends HttpServlet
2)  {
3)     public void doGet(..)...
4)     {
5)        PrintWriter out = resp.getWriter();
6)        out.println(10/0);
7)     }
8)  }
```

error.jsp:

```
<h1>Hello your provided input is invalid...plz provide valid input</h1>
```

error404.jsp:

```
<h1>Hello Stupid..Plz cross check your url and send valid url</h1>
```

web.xml:

```
1)  <web-app>
2)
3)  <servlet>
4)   <servlet-name>FirstSevlet</servlet-name>
```

```
5)    <servlet-class>FirstSevlet</servlet-class>
6)   </servlet>
7)
8)   <servlet-mapping>
9)    <servlet-name>FirstSevlet</servlet-name>
10)  <url-pattern>/test</url-pattern>
11) </servlet-mapping>
12)
13) <error-page>
14)   <exception-type>java.lang.ArithmeticException</exception-type>
15)   <location>/error.jsp</location>
16) </error-page>
17)
18) <error-page>
19)   <error-code>404</error-code>
20)   <location>/error404.jsp</location>
21) </error-page>
22) </web-app>
```

```
advapps2H
   |-error.jsp
   |-error404.jsp
   |-WEB-INF
      |-web.xml
      |-classes
            |-FirstServlet.class
```

If we are sending the request to the FirstServlet then instead of getting ArithmeticException,we will get error.jsp page response.

If we are sending the request with invalid url-pattern then instead of 404 status code,we will get error404.jsp page response.

Sending Error Code Programatically:

We can set error code programatically. HttpServletResponse interface defines the following method for this.

public void sendError(int errorCode)

Eg: resp.sendError(401);

Demo Program 2 for demonistrating error pages:

login.html:

```
1.  <h1>
2.   <form action ="/advapps2I/test" >
```

```
3.  Enter Name: <input type="text" name="uname"><br>
4.  <input type="submit" value="Login"/>
5.  </form></h1>
```

FirstServlet.java:

```
1.  import javax.servlet.*;
2.  import javax.servlet.http.*;
3.  import java.io.*;
4.  import java.util.*;
5.  public class FirstServlet extends HttpServlet
6.  {
7.     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
8.     {
9.        PrintWriter out = resp.getWriter();
10.       String name= req.getParameter("uname");
11.       if(name.equals("durga"))
12.       {
13.          out.println("<h1>your authentication is correct...you can avail the facilities</h1>");
14.       }
15.       else
16.       {
17.          resp.sendError(401);
18.       }
19.    }
20. }
```

error401.jsp:

```
1.  <h1>Your authentication is failed....plz provide valid credentials...<br/>
2.  <a href="/advapps2I/login.html">Click here to login once again</a></h1><br>
```

web.xml:

```
1.  <web-app>
2.
3.     <error-page>
4.        <error-code>401</error-code>
5.        <location>/error401.jsp</location>
6.     </error-page>
7.
8.  <servlet>
9.    <servlet-name>FirstServlet</servlet-name>
10.   <servlet-class>FirstServlet</servlet-class>
11. </servlet>
12.
13. <servlet-mapping>
14.   <servlet-name>FirstServlet</servlet-name>
```

```
15.  <url-pattern>/test</url-pattern>
16. </servlet-mapping>
17.
18. </web-app>
```

advapps2I
|-login.html
|-error401.jsp
|-WEB-INF
|-web.xml
|-classes
|-FirstServlet.class

Case-1:
We can configure <error-page> either based on exception type or based on error code but not both simultaneously in the same <error-page> tag.

Eg:

```
1)  <error-page>
2)    <exception-type>java.lang.AE</exception-type>
3)    <error-code>404</error-code>
4)    <location>/error.jsp</location>
5)  </error-page>
```

It is invalid

Case-2:
The <exception-type> tag value should be fully qualified name of exception.

```
1)  <error-page>
2)    <exception-type>IOException</exception-type>
3)    <location>/error.jsp</location>
4)  </error-page>
```

It is invalid

Case-3:
The <location> value must be compulsary starts with "/",otherwise application won't be deployed.

```
1)  <error-page>
2)    <error-code>404</error-code>
3)    <location>error.jsp</location>
4)  </error-page>
```

It is invalid

<mime-mapping>:

We can use <mime-mapping> to map file extension with the corresponding <mime-type>.

```
1)  <mime-mapping>
2)     <extension>durga</extension>
3)     <mime-type>application/pdf</mime-type>
4)  </mime-mapping>
```

<mime-mapping> is the direct child tag of <web-app> and hence we can take anywhere.

war File

Objective:

1. Explain the purpose of war file?
2. Describe the contents of war file?
3. Explain the process of creation?

war (Web Archieve) provides a convenient way to store resources of web appication in a single component.
It is a compressed file(zip file) represents total web application which may contains Servlets, JSPs, XML Files, HTML Pages, JavaScript Files, CSS Files etc..

The main advantage of maintaining web application in the form of war file is Project delivery, Project transportation and deployment will become easy.

Servlet specification defined a standard structure for the war file and every web server can provide support for that war file.

Various Commands:

1. Creation of war File (zip file):

```
jar -cvf mywebapp.war  *
```

2. Extraction of war File (unzip operation)

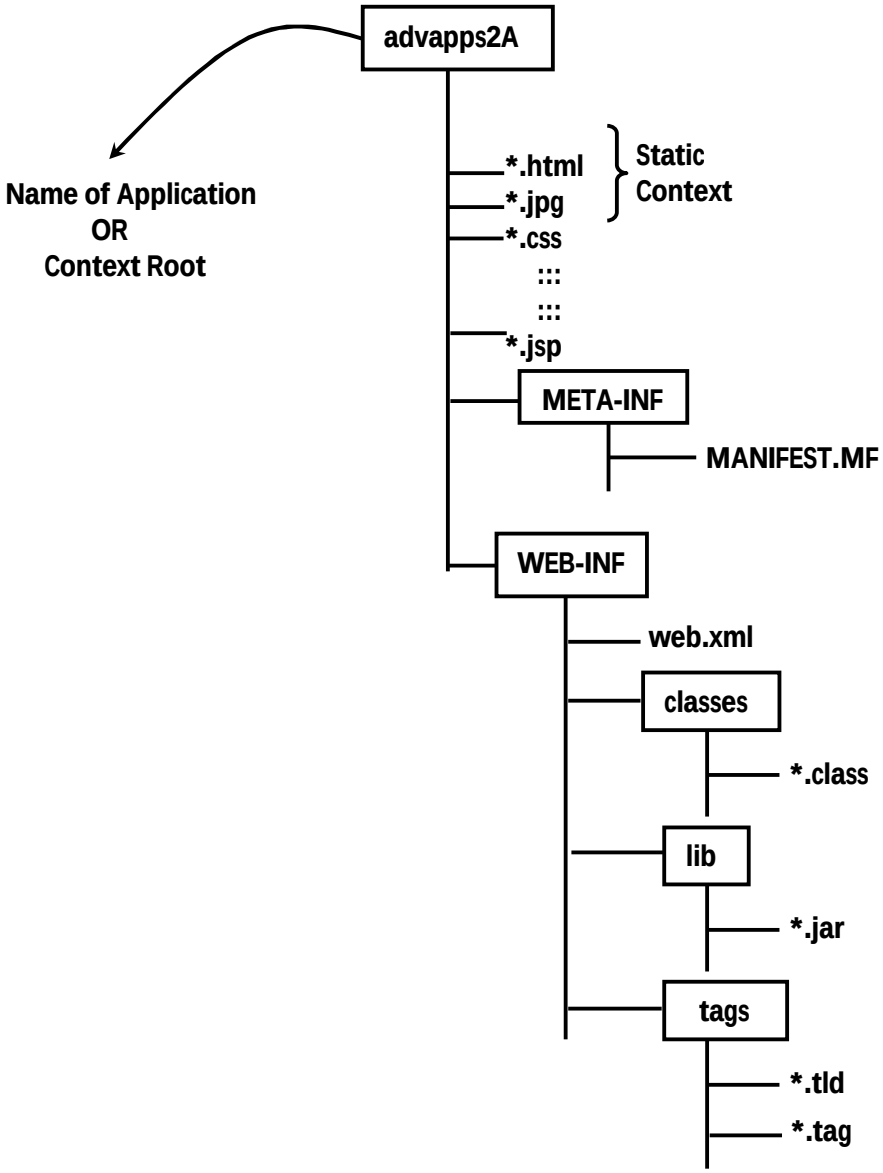
```
jar -xvf mywebapp.war
```

3. Display Table of contents of war File

```
jar -tvf mywebapp.war
```

http://localhost:7777/wardemo/login.html

Structure of war File:



Every war file should compulsary contains META-INF folder, which should contain MANIFEST.MF. i.e META-INF folder and MANIFEST.MF are mandatory for every war file.

META-INF folder contains security related resources like signature files,digital certificates, license agreements etc which are mandatory for maintaining the web application.

MANIFEST.MF:

If there is any jar file which is common to several web applications,then it is not recommended to place that jar file at application level.We have to place that jar file at some common location outside of web application and we can define that path in MANIFEST.MF.

ie MANIFEST.MF contains the classpath of common jar files stored outside of web application.i.e The library dependencies of web application,we can define in this file only.

The resources present inside META-INF and WEB-INF folders cannot be accessed directly. if we are trying to access then we will get 404 status code.

http://localhost:7777/wardemo/META-INF/MANIFEST.MF
http://localhost:7777/wardemo/WEB-INF/web.xml

jar vs war vs ear:

1.jar(Java archieve):
It contains a group of .class files

2. war(web archieve)
It represents one web application that contains servlets,jsps,html files...
3. ear(enterprise archieve):
It represents an enterprise application which contains servlets,jsps,ejbs,jms components etc..

UNIT-3: The Web Container Model

1. ServletContext
2. Servlet Scopes and Attributes
3. RequestDispatcher
4. Filters
5. Wrappers

1. ServletContext:

Objective: For the servlet context parameters
1. Write Servlet code to access parameters
2. Create Deployment Descriptor elements for the initialization parameters

For every servlet web container will create one ServletConfig object to hold servlet level configuration information. By using this config object servlet can get its configuration information like logical name of servlet, initialization parameters etc

Similarly for every web application , web container creates one ServletContext object to maintain application level configuration information.By using ServletContext object, servlet can get application level configuration information like context parameters,RequestDispatcher etc..

ServletConfig is per Servlet where as ServletContext is per web application
If initialization parameters are common for several servlets then it is not recommended to declare those parameters at servlet level.We have to declare those parameters at application level by using <context-param> tag.

```
1)  <web-app>
2)    <context-param>
3)        <param-name>user</param-name>
4)        <param-value>scott</param-value>
5)    </context-param>
6)    ...
7)  </web-app>
```

We can declare any number of context parameters but seperate <context-param> tag for every parameter.

<context-param> is the child tag of <web-app> and hence within <web-app> we can declare anywhere.

Within the servlet we can access context initialization parameters by using ServletContext object.

We can get ServletContext object by using getServletContext() method of ServletConfig interface.

```
ServletContext context=getServletContext();
```
(or)

ServletConfig config=getServletConfig();
ServletContext context=config.getServletContext();
ServletContext interface defines the following methods for accessing context initialization parameters.

public String getInitParameter(String pname)
public Enumeration getInitParameterNames()

Demo Program for Servlet Context Parameters:

web.xml:

```
1)  <web-app>
2)  <context-param>
3)     <param-name>PhoneNumber</param-name>
4)     <param-value>9292929292</param-value>
5)  </context-param>
6)
7)  <context-param>
8)     <param-name>mailid</param-name>
9)     <param-value>support@durgasoft.com</param-value>
10) </context-param>
11)
12) <servlet>
13)  <servlet-name>DemoServlet</servlet-name>
14)  <servlet-class>InitializeParameterDemoServlet</servlet-class>
15) </servlet>
16)
17) <servlet-mapping>
18)  <servlet-name>DemoServlet</servlet-name>
19)  <url-pattern>/test</url-pattern>
20) </servlet-mapping>
21)
22) </web-app>
```

InitializeParameterDemoServlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import java.util.*;
5)  public class InitializeParameterDemoServlet extends HttpServlet
6)  {
7)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
8)     {
9)        PrintWriter out = resp.getWriter();
10)       out.println("<center><h1>Intialization Parameters</h1></center><hr>");
11)       ServletContext context = getServletContext();
```

```
12)      Enumeration e =context.getInitParameterNames();
13)      out.println("<table border=2><tr><th>Parameter Name</th><th>Parameter  Value</t
    h></tr>");
14)      while (e.hasMoreElements())
15)      {
16)         String pname = (String)e.nextElement();
17)         String pvalue = context.getInitParameter(pname);
18)         out.println("<tr><td>"+pname+"</td><td>"+pvalue+"</td></tr>");
19)
20)      }
21)      out.println("</table>");
22)      out.println("</body></html>");
23)   }
24) }
```

Note: By default initialization parameter means Servlet Initialization Parameter but not context initialization parameter.
We can access Servlet Initialization Parameters in the following ways..

String value=getInitParamter("user");
String value=getServletConfig().getInitParamter("user");

We can access Context Initialization Parameters as follows

String value=getServletContext().getInitParameter("user");
String value=getServletConfig().getServletContext().getInitParameter("user");

Comparison between Servlet and Context Initialization Parameters:

Properties	Servlet init-param	Context init-param
1) Deployment Descriptor Declaration	By using <init-param> within Servlet <Servlet> <init-param> <param-name> <param-value> </init-param> </Servlet>	By using <context-param> within <web-app> <web-app> <context-param> <param-name> <param-value> </context-param> </web-app>
2) Servlet Code to access the Parameters	String value = getInitParameter("pname"); OR String value = getServletConfig().getInitParameter("pname")	String value = getServletContext().getInitParameter("pn"); OR String value = getServletConfig().getServletContext().getInitParameter("pname");
3) Availability (Scope)	Available only for a particular Servlet in which <init-param> is declared	Available for all Servlets and JSP's within the Web Application

Note: whether servlet or context, all initialization parameters are deployment time constants.From the servlet we can get these values but we cannot set. i.e We have only getter methods but not setter methods.

Differences between ServletConfig and ServletContext:

ServletConfig:

1) For every Sevlet Web Container creates one ServletConfig Object.
2) ServletConfig Object will be created at the time of Servlet Object Creation and destroyed at the time of Servlet Object Destruction.
3) Web Container hand-over Config Object to the Servlet as an Argument to init() Method.
4) Within the Servlet we can get its Config Object as follows
ServletConfig config = getServletConfig();
5) It is not Object of javax.servlet.ServletConfig(I). It is the Object of its Implementation Class provided by Web Server Vendor.
6) By using Config Object, Servlet can get its Configuration Information like Logical Name of the Servlet, Initialization Parameters etc.. by using the following Methods.

- getServetName()
- getInitParameter()
- getInitParameterNames()
- getServletContext()

ServletContext:

1) For every Web Application, Web Container creates one ServletContext Object to hold Application Level Configuration Information.
2) Context Object will be created at the time of Application Deployment and destroyed at the time of Application Undeployment.
3) Within the Servlet we can get Context Object as follows:
ServletContext context = config.getServletContext();
ServletContext context = getServletContext();
4) It is not Object of javax.servlet.ServletContext(I). It is the Object of its Implementation Class provided by Web Server Vendor.
5) On the ServletContext Object we can call the following Methods

- getRequestDispatcher()
- getInitParameter()
- getAttribute()
- getServletInfo()
- getResourcePaths()
- getContextPath()
- log()
- getMajorVersion()
- getMinorVersion()

*******FAQs:**

1. What is the difference between ServletConfig and ServletContext?
2. What is the difference between Servlet Initialization Parameters and Servlet Context Parameters?

Servlet Scopes and Attributes

Objective:

For the fundamental servlet scopes (request,session,context)

1. Write Servlet Code to add,retrieve and remove attributes?
2.For the given scenario identify proper scope?
3.Identify multi threading issues associated with each scope?

There are 3 Types of Parameters are possible.

1. Form Parameters
2. Servlet Initialization Parameters
3. Context Initialization Parameters

These parameters are read-only. i.e from the servlet we can perform only read operation and we cannot modify, remove values based on our requirement. Hence Parameters concept is not useful for sharing data between components of web application.

To resolve this problem we should go for attributes concept.Based on our requirement we can create a new attribute,we can modify the value and we can remove existing attribute. Hence attributes concept is best suitable for sharing data between components of web application.
Based on our requirement,we have to store attributes in the proper scope.
There are 3 scopes are possible for the servlets.

1. Request Scope
2. Session Scope
3. Application/Context Scope

1. Request Scope:

1. Request scope is maintained by either ServletRequest object or HttpSerlvetRequest object.

2. Request scope will start at the time of request object creation(i.e just before calling service() method)and ends at the time of request object destruction(i.e just after completing service() method)

3.The data stored in request scope is available for all components which are processing that request.

4. ServletRequest interface defines the following methods for attribute management in the Request scope

1.public void setAttribute(String name,Object value)
To add an attribute.
If the specified attribute is already avaialble then the old value is replaced with new value.

2.public Object getAttribute(String name)
Returns the value associated with the specified attribute.
If the specified attribute is not available then this method returns null.
3. public void removeAttribute(String name)
4. public Enumeration getAttributeNames()

Example:

The most common application area where we can use Request scoped attributes is RequestDispatcher Mechanism

For every request a seperate new request object will be created,which can be accessed by only current thread.Other threads are not allowed to access request scoped attributes.Hence request scoped attributes are always thread-safe.

Session Scope:

Session scope is maintained by HttpSession object.
Session Scope will start at the time of Session object creation.

```
HttpSession session=req.getSession();
```

Session scope ends at the time of Session object destruction(ie at the time of either logout or timeout)

```
session.invalidate();
```

Session Object Creation
HttpSession session=req.getSession();

session.invalidate();

The information stored in the session scope is available for all the components which are participating in that session.

HttpSession interface defines the following methods for attribute management in session scope

1. public void setAttribute(String name,Object value)
2. public Object getAttribute(String name)
3. public void removeAttribute(String name)
4. public Enumeration getAttributeNames()

***Note:**
Once session expired, we are not allowed to call these methods otherwise we will get RuntimeException saying IllegalStateException.

Example: login information should be available for total session. Hence we have to store this information in the session scope.

within the same session we can send multiple requests simultaneously by opening multiple tabs. Hence session object can be accessed by mutilple threads simultaneously and hence session scoped attributes are not Thread Safe.

Application Scope:

Application scope is maintained by ServletContext object.
This scope will start at the time of context object creation. ie at the time of application deployment or server startup.

Application scope ends at the time of context object destruction. i.e at the time of application undeployment or server shutdown.

The data stored in application scope will be available to all the components of web application irrespective of request and end user.

ServletContext interface defines the following methods for attribute management in the application scope...

1. public void setAttribute(String name,Object value)
2. public Object getAttribute(String name)
3. public void removeAttribute(String name)
4. public Enumeration getAttributeNames()

ServletContext object can be accessed simultaneously by multiple threads and hence context scoped attributes are not thread safe.

Instance and static variables can be accessed by multiple threads simultaneously and hence these are not thread safe.
For every Thread a seperate copy of local variables will be created and hence local variable can be accessed by only current thread. Hence local variables are Thread Safe.

Table:

Member	Is Thread Safe?
Request Scope Attribute	Yes
Session Scope Attribute	No
Context Scope Attribute	No
Instance Variables	No
Static Variables	No
Local Variables	Yes

Parameters are key-value pairs where both key and value are String objects. Hence at the time of retrieval it is not required to perform any typecasting.We can assign directly parameter value to the String type variable.

Eg:
String user=req.getParameter("user");
String user=getInitParameter("user");

Attributes are also key-value pairs. But keys are String type and values can be any type. Hence at the time of retrieval compulsary we should perform type casting.

String user=req.getAttribute("user");
CE: incompatible types
found: Object
required: String

String user=(String)req.getAttribute("user");

Q. To access the value of request scoped attribute user,which of the following is valid way?

1.String user=req.getParameter("user");
2.String user=req.getInitParameter("user");
3.String user=getInitParameter("user");
4.String user=req.getAttribute("user");
5.String user=(String)req.getAttribute("user"); ✔

Differences b/w Parameters and Attributes:

Parameter	Attribute
1) Parameters are Read Only i.e. within the Servlet we can perform Read Operation but we can't modify their Values i.e., we have only getters()'s but not setters()'s.	1) Based on our Requirement we can get and set the Attributes i.e., these are not Read only and we have both getters()'s and setters()'s.
2) Parameters are Deployment Time Constants.	2) Attributes are not Deploy time Constants.
3) Parameters are Key - Value Pairs and both Key and Value are String Objects only. Keys ---- String Values ---- String	3) Attributes are Key - Value Pairs where Key is String but Value can be any Type of Object. Keys ---- String Value ---- Object
4) At the time of retreival it is not require to perform Type casting.	4) At the time of retrieval we should perform Type casting.

Q1. Demo Program to display hit count(number of requests) of web application?

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
```

```
3)  import java.io.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test")
6)  public class HitCountDemo extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
9)     {
10)       PrintWriter out = resp.getWriter();
11)       ServletContext context = getServletContext();
12)       Integer count = (Integer)context.getAttribute("hitcount");
13)       if(count == null)
14)       {
15)          count = 1;
16)       }
17)       else
18)       {
19)          count++;
20)       }
21)       context.setAttribute("hitcount",count);
22)       out.println("<h1>The number of requests is:"+count+"</h1>");
23)    }
24) }
```

Q2. Demo Program to display the number of users login in our application.

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test")
6)  public class UsersCount extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
9)     {
10)       PrintWriter out = resp.getWriter();
11)       ServletContext context = getServletContext();
12)       Integer count = (Integer)context.getAttribute("usercount");
13)       HttpSession session = req.getSession();
14)       if(session.isNew())
15)       {
16)          if(count == null)
17)          {
18)             count = 1;
19)          }
20)          else
21)          {
22)             count++;
```

```
23)         }
24)         context.setAttribute("usercount",count);
25)       }
26)       out.println("<h1>The no of users login into our application :"+count+"</h1>");
27)    }
28) }
```

Q. Demo Program to display number of requests in the current session?

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test")
6)  public class SessionHitCountDemo extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
9)     {
10)       PrintWriter out = resp.getWriter();
11)       HttpSession session = req.getSession();
12)       Integer count = (Integer)session.getAttribute("hitcount");
13)       if(count == null)
14)       {
15)          count =1;
16)       }
17)       else
18)       {
19)          count++;
20)       }
21)       session.setAttribute("hitcount",count);
22)       out.println("<h1>The number of requests in the current session is:"+count+"</h1>");
23)    }
24) }
```

Q. Write a program to display all attributes information present in application scope?

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import java.util.*;
5)  import javax.servlet.annotation.*;
6)  @WebServlet("/test")
7)  public class ContextAttributeDemo extends HttpServlet
8)  {
9)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
10)    {
11)       PrintWriter out = resp.getWriter();
```

```
12)     out.println("<h1>Context Attributes</h1>");
13)     ServletContext context=getServletContext();
14)     context.setAttribute("durga","SCWCD");
15)     Enumeration e = context.getAttributeNames();
16)
17)     while(e.hasMoreElements())
18)     {
19)        String name= (String)e.nextElement();
20)        Object value = context.getAttribute(name);
21)        out.println(name+"....."+value+"<br>");
22)     }
23)   }
24) }
```

Note: In application scope web container will add some attributes for its internal purpose.

RequestDispatcher

Objective:

1. Describe RequestDispatcher Mechanism?
2. Write Servlet code to create RequestDispatcher?
3. Write Servlet code to forward or include target resource?
4. Identify & Describe attributes added by web container while forwarding and including?

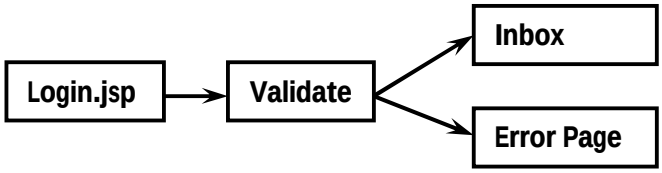
It is not recommended to define total functionality in a single component. It has several serious disadvantages.

1. Without effecting remaining logic we cannot modify any code. Hence enhancement will become difficult and maintainability of the application will be down.

2. It does not promote reusability of the code.

Total Servlet

We can resolve these problems by maintaining a seperate component for each task.



The main advantages of this approach are

1. Without effecting remaining components we can modify any component. Hence enhancement will become very easy and improves maintainability of the application.

2. It promotes reusability.

Eg: Where ever validation is required, we can reuse the same ValidateServlet without rewriting.

If total functionality is distributed across multiple components then these components have to communicate with each other to provide response to end user. We can achieve this communication by using RequestDispatcher.

Hence the main purpose of RequestDispatcher is to dispatch our request from one component to another component.

Eg: After validation ValidateServlet has to forward the request to inbox.jsp.For this we can use RequestDispatcher.

Servlet code for Getting RequestDispatcher:

We can get RequestDispatcher either by using ServletRequest object or by using ServletContext object.

1. By ServletRequest object:

ServletRequest interface defines the following method for this purpose.

public RequestDispatcher getRequestDispatcher(String targetResouce)
The target Resource can be specified either by absolute path or by relative path.

Eg:

```
RD rd = req.getRequestDispatcher("/test2");
RD rd = req.getRequestDispatcher("test2");
```

If the target resouce is not available then we will get 404 status code saying requested resource is not available.

By ServletContext Object:

ServletContext interface defines the following methods for getting RequestDispatcher.

1. public RequestDispatcher getRequestDispatcher(String targetResource)

The target Resource should be specified by only absolute path. If we are using relative path(i.e the path not starts with "/")then we will get Runtime Exception saying IllegalArgumentException.

Eg:
RD rd = context.getRequestDispatcher("test2");==>RE:IAE
RD rd = context.getRequestDispatcher("/test2");
If the target resource is not available then we will get 404 status code saying requested resource is not available.

2. public RequestDispatcher getNamedDispatcher(String servletName)

The argument represents the value associated with the <servlet-name> tag in web.xml. i.e it represents logical name of the servlet.

If url-pattern is not available then we can use this method.

If the specified servlet is not available then we will get null. On that null if we are trying to call any method then we will get NullPointerException.

Eg:
RD rd = context.getNamedDispatcher("FirstServlet");

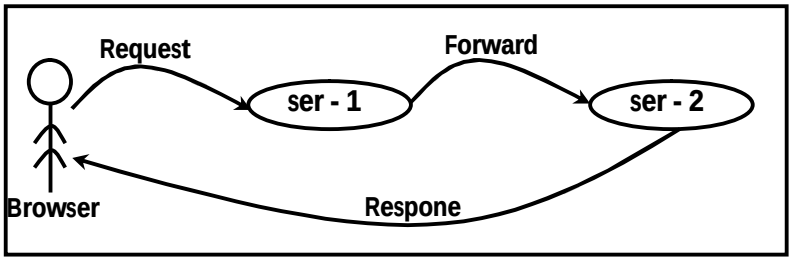
Difference b/w getting RequestDispatcher by ServletRequest and by ServletContext:

RD From Request Object	RD From Context Object
1) We can get RD only based on URL Pattern but not based on Servlet Name.	1) We can get RD based on either URL Pattern OR based on Servlet Name.
2) RD rd = req.getRD("/test2"); ✔ RD rd = req.getRD("test2"); ✔ We can use either Relative Path OR Absolute Path	2) RD rd = context.getRD("/test2"); ✔ RD rd = context.getRD("test2"); ✕ We should use only Absolute Path but not Relative Path, otherwise we will get IlleagalArgumentException
3) By using this RD, we can communicate only within the Web Application i.e., Cross Context Communication is not possible.	3) By using this RD, we can communicate either within the Web Application OR outside of the Web Application i.e., Cross Context Communication is possible.

Methods of RequestDispatcher:

Once we got RD, we can call the following 2 methods on this object.
1. public void forward(SR req,SR resp)throws SE,IOE
2. public void include(SR req,SR resp)throws SE,IOE

Forward Mechanism:



If the FirstServlet is responsible for some preliminary processing & 2nd Servlet is responsible to provide required response, then we should go for forward mechanism.

Demo Program 1 to forward:

FirstServlet.java

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test1")
6)  public class FirstServlet extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
9)     {
10)       PrintWriter out=resp.getWriter();
11)       out.println("<h1>This is First Servlet</h1>");
12)       RequestDispatcher rd = req.getRequestDispatcher("/test2");
13)       rd.forward(req,resp);
14)    }
15) }
```

SecondServlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test2")
6)  public class SecondServlet extends HttpServlet
7)  {
8)
9)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
10)    {
11)       PrintWriter out=resp.getWriter();
12)       out.println("<h1>This is Second Servlet</h1>");
13)
14)    }
15) }
```

If we are sending the request to FirstServlet then it will forward request to SecondServlet and ServletServlet is responsible to provide required response.

Demo Program 2 to forward:

login.html:

```
1)  <h1> This is forward demo</h1>
2)  <form action = "/advapps3D/test1" >
3)  Enter Name :<input type=text name=uname><br>
4)  Enter Password :<input type=text name=pwd><br>
5)  <input type=submit>
6)  </form>
```

ValidateServelt.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test1")
6)  public class ValidateServlet extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
9)     {
10)       String name = req.getParameter("uname");
11)       String pwd = req.getParameter("pwd");
12)       if(name.equals("Durga") && pwd.equals("scwcd"))
13)       {
14)          ServletContext context=getServletContext();
15)          RequestDispatcher rd =context.getRequestDispatcher("/inbox.jsp");
16)          rd.forward(req,resp);
17)       }
18)       else
19)       {
20)          RequestDispatcher rd =req.getRequestDispatcher("/error.jsp");
21)          rd.forward(req,resp);
22)       }
23)
24)    }
25) }
```

inbox.jsp:

```
<h1>This is inbox page you can get all mail services</h1>
```

error.jsp:

```
<h1>This is error page your credentials are invalid please login again here
<a href="/advapps3D/login.html">LOGIN</a></h1>
```

Case-1:
Just before forwarding the request,the response object will be cleared automatically by web container.Hence if any response added by FirstServlet won't be displayed to the end user.

Case-2:
In Forward Mechanism the same request object will be forwarded to the SecondServlet.Hence information sharing b/w the components is possible in the form of request scoped attributes.

FirstServlet:
req.setAttribute("count",10);

SecondServlet:
Object o = req.getAttribute("count");

Case-3:
In forward mechanism, SecondServlet has complete control on the response object.It can change response headers which are added by FirstServlet like content-type etc..

Case4:
After commiting the response, we are not allowed to forward the request. Otherwise we will get Runtime Exception saying IllegalStateException.

```
1)  public class  HelloServlet extends HttpServlet
2)  {
3)     public void doGet(..)...
4)     {
5)        PrintWriter out=resp.getWriter();
6)        out.println("This is required response");
7)        out.flush();//commiting the response
8)        RD rd = req.getRD("/test2");
9)        rd.forward(req,resp);//RE: ISE
10)    }
11) }
```

Note: Tomcat does not implement this feature.In the case of Tomcat, FirstServlet response will be displayed to the end user.

case-5:
Recursive forward call is always a RuntimeException saying StackOverflowError.

FS:
RD rd =req.getRD("/test2");
rd.forward(req,resp);

SS:
RD rd =req.getRD("/test1");
rd.forward(req,resp);

case-6:

After forward call the control comes back to execute remaining statements.In the remaining statements if we are trying to write anything to response, then these statements will be ignored by web container.

In the remaining statements if any exception occurs then that exception information will be displayed to the end user but not SecondServlet response.

If all remaining statements executed successfully then only SecondServlet response will be displayed to the end user.

```
1)  public class FirstServlet extends HttpServlet
2)  {
3)     public void doGet(..)...
4)     {
5)        PrintWriter out=resp.getWriter();
6)        RequestDispatcher rd=req.getRequestDispatcher("/test2");
7)        rd.forward(req,resp);
8)        System.out.println("After forward control comes back");// it will printed in the server
    console
9)        out.println("Hello this is FirstServlet again");//This line ignored by web container
10)       System.out.println(10/0);// AE information will be displayed to the end user instead o
    f secondservlet response
11)    }
12) }
```

Attributes added by web container while forwarding the request:

While forwarding the request from one servlet to another servlet,web container will add some attributes in the request scope to make original request information available to the Second Servlet.

SecondServlet will use these attributes to get original request information.

Web container will add the following attributes in request scope.

javax.servlet.forward.request_uri
javax.servlet.forward.context_path
javax.servlet.forward.servlet_path
javax.servlet.forward.path_info
javax.servlet.forward.query_string

Demo Program:

FirstServlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
```

```
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test1")
6)  public class FirstServlet extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
9)     {
10)       req.setAttribute("durga","java");
11)       ServletContext context=getServletContext();
12)       RequestDispatcher rd = req.getRequestDispatcher("/test2");
13)       rd.forward(req,resp);
14)    }
15) }
```

ForwardAttributeDemo.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import java.util.*;
5)  public class ForwardAttributeDemo extends HttpServlet
6)  {
7)
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws
    ServletException,IOException
9)     {
10)       PrintWriter out = resp.getWriter();
11)       out.println("<h1>Forward Request Attributes</h1>");
12)       Enumeration e = req.getAttributeNames();
13)       while(e.hasMoreElements())
14)       {
15)          String name= (String)e.nextElement();
16)          Object value = req.getAttribute(name);
17)          out.println(name+"....."+value+"<br>");
18)       }
19)    }
20) }
```

http://localhost:7777/advapps3F/test1/durga/software?user=durga&pwd=anushka

Output

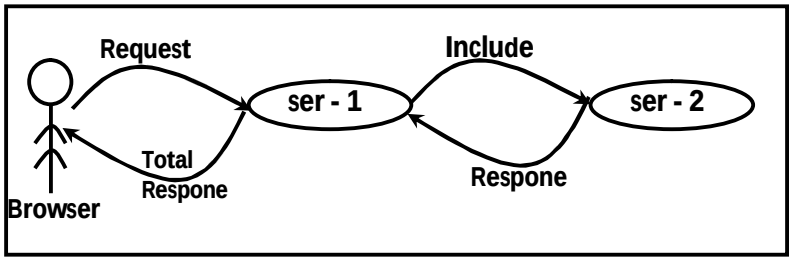
```
javax.servlet.forward.request_uri...../advapps3F/test1/durga/software
javax.servlet.forward.context_path...../advapps3F
javax.servlet.forward.servlet_path...../test1
javax.servlet.forward.path_info...../durga/software
javax.servlet.forward.query_string.....user=durga&pwd=anushka
durga.....java
```

http://localhost:7777/advapps3F/test2

If we are sending the request directly to the ForwardAttributeDemo servlet then we won't get any attributes.

Note:If we are getting RequestDispatcher by getNamedDispatcher() method then web container won't add any attributes in the request scope while forwarding the request.

Include Mechanism:



We can use include mechanism to include the response of other resources in the current response.

Inlcude mechanism is best suitable for including the banner information like copyright, logo etc..

The servlet which is getting request initially is responsible to provide response.

After commiting the response we can perform include() call, but we cannot perform forward call.

In the include SecondServlet does not have complete control on the response object.It is not allowed to change response headers. If it is trying to perform any changes then these things will be ignored by web container.

Demo Program for include:

FirstServlet.java

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test1")
6)  public class FirstServlet extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
9)     {
10)       PrintWriter out=resp.getWriter();
11)       out.println("<h1>Hello This is FirstServlet</h1>");
12)       RequestDispatcher rd=req.getRequestDispatcher("/test2");
13)       rd.include(req,resp);
14)       out.println("<h1>Hi This is First Servlet again</h1>");
15)    }
16) }
```

SecondServlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test")
6)  public class SecondServlet extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
9)     {
10)       PrintWriter out=resp.getWriter();
11)       out.println("<h1>This is Second Servlet</h1>");
12)    }
13) }
```

If we are sending the request to FirstServlet then the output is:

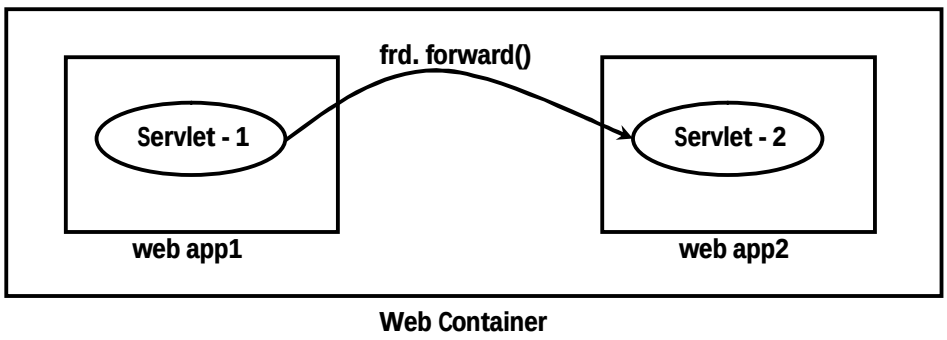
Hello This is FirstServlet
This is Second Servlet
Hi This is First Servlet again

Attributes added by web container in request scope while performing include call:

javax.servlet.include.request_uri
javax.servlet.include.context_path
javax.servlet.include.servlet_path
javax.servlet.include.path_info
javax.servlet.include.query_string

Note: If we are getting RequestDispatcher by getNamedDispatcher() method then web container won't add these attributes...

Foreign RequestDispatcher:



If we want to communicate with the resouces of other applications but within the same server then we should go for FRD.

FirstServlet.java (advapps3H):

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test1")
6)  public class FirstServlet extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
9)     {
10)       ServletContext context=getServletContext();
11)       ServletContext fc=context.getContext("/advapps3I");
12)
13)       RequestDispatcher rd=fc.getRequestDispatcher("/test2");
14)       rd.forward(req,resp);
15)    }
16) }
```

SecondServlet.java (advapps3I):

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test2")
6)  public class SecondServlet extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
9)     {
10)       PrintWriter out=resp.getWriter();
11)       out.println("<h1>This is Second Servlet..You can access by Foreign Request Dispatcher
    </h1>");
12)    }
13) }
```

http://localhost:7777/advapps3H/test1

Note:

1.RequestDispatcher mechanism will work with in the same server. Hence both applications should be deployed in the same server.

2. Most of the web servers wont provide support for cross context communication due to security reasons. In this case we will get NullPointerException.

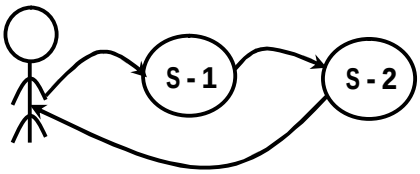
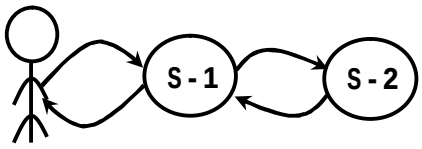
To provide support for cross context communication we have to perform configuration changes at server level.

In Tomcat we have to add the following in context.xml(Tomcat/conf folder)

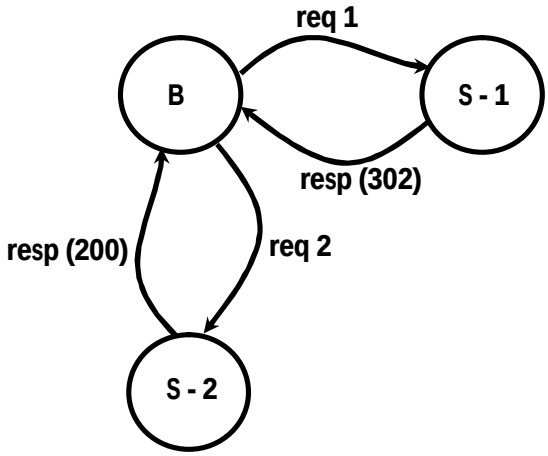
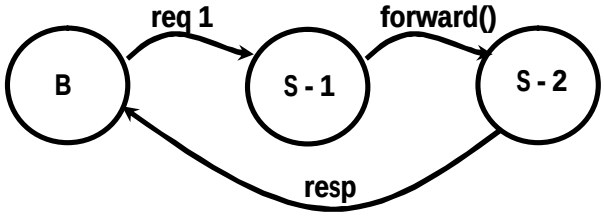
```
<Context crossContext="true">
```

Note: After commiting the response,we are not allowed to perform sendRedirect() and forward() calls.Otherwise we will get RE Saying IllegalStateException.

Differences Between forward() and include():

forward()	include()
	
1) If FirstServlet is responsible for some preliminary processing and SecondServlet is responsible to provide complete Response then we should go for Forward Mechanism.	1) If we want to Include the Response of other Servlets in the Current Servlet Response at Runtime then we should go for Include Mechanism.
2) In the case of Forward, ForwardedServlet (S - 2) is responsible to provide complete Response.	2) In the case of Include, IncludingServlet (S - 1) is responsible to provide Response.
3) Within the same Servlet, we can call forward() only once, mostly as the last Statement.	3) Within the same Servlet, we can call include() any Number of times and there are no restrictions.
4) In the case of Forward, ForwardedServlet (S - 2) having complete Control on the Response Object. It is allowed to change Response Headers also.	4) In the case of Include, IncludingServlet (S - 1) having complete Control on the Response Object. It is allowed to change Response Headers also.
5) After committing the Response, we are not allowed to perform Forward, otherwise we will get RE: IlleagalStateException.	5) After committing the Response, we are allowed to perform Include.
6) Forward Mechanism can be used frequently in Servlets because it is associated with processing.	6) Include Mechanism can be used frequently in JSP's because it is associated with Presentation Logic.
7) Eg: After validating User, the Request should be forwarded to Inbox Page.	7) Eg: We required to Include Header Information and Footer Information in the Current Response.

Differences Between forward() and sendRedirect():

sendRedirect()	forward()
 1) Redirection Mechanism will work at Client Side. Hence Client aware of which Servlet is providing the required Response.	 1) Forward Mechanism will work at Server Side and it is not visible to the Client. Hence Client does not aware of which Servlet is providing required Response.
2) Redirection will work either within the Server OR outside of Server.	2) Forward will work only within the Server and won't work outside of Server.
3) It is the Best Choice if we want to communicate outside of Server.	3) It is the Best Choice if we want to communicate within Server.
4) A separate New Request Object will be created in Redirection. Hence Information sharing between the Components is not possible.	4) The same Request Object will be forwarded to the SecondServlet and hence Information sharing between the Components is possible in the form of Request scoped Attributes.
5) In this Approach an extra trip is required to the Client Side. Hence Network Traffic increases and creates Performance Problems.	5) No extra trip is required to the Client and Hence there are no Network Traffic and Performance Problems.
6) By using HttpServletResponse Object we can implement sendRedirection. resp.sendRedirection()	6) By using RequestDispatcher Object we can implement Forward Mechanism. rd.forward()

FAQs:

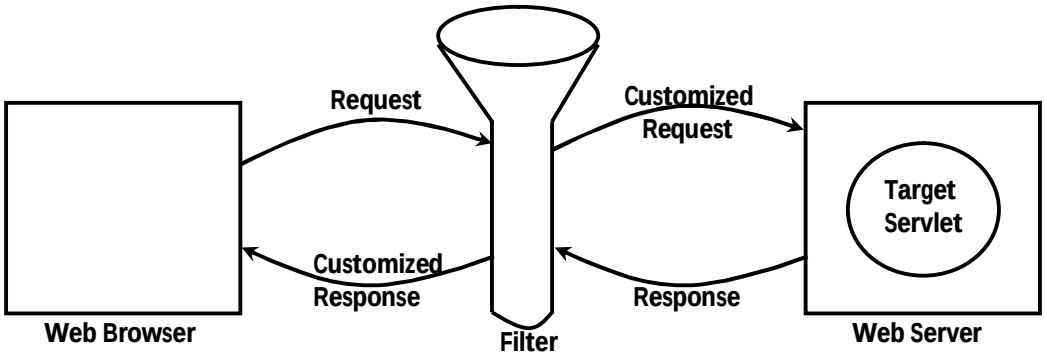
1. What is the purpose of RequestDispatcher?
2. In How many ways we can get RD?
3. What is the difference b/w getting RD from request and context objects?
4. what are differences b/w forward() and include()?
5. what are differences b/w forward() and sendRedirect()?
6. What is Foreign RequestDispatcher and how we can get it?
7. What are the attributes added by web container while forwarding and including the request.
8. What is the purpose of these attributes and explain meaning?

Filters & Wrappers

Objectives:

1. Describe the web container's Request Processing Model?
2. Write & Configure a Filter?
3. Create a Request & Response Wrapper for the given problem?
4.Describe how to apply a Filter or wrapper?

Filters can be used for pre processing of request and post processing of request before they reach target resource in the web application.



The most common application areas of Filters are logging,security checks,altering request information,compressing response,encryption of response,authentication etc...

Filters concept introduced in Servlet 2.3 Version.

Filter API

Filter API contains the following 3 interfaces.

1. Filter
2. FilterConfig
3. FilterChain

1. Filter(I):

Every Filter in Java has to implement Filter interface either directly or indirectly.

Filter interface defines the following 3 methods

1.init():

public void init(FilterConfig config)throws ServletException
This method will be executed only once to perform initialization activities.

2. destroy()

public void destroy()

This method will be executed only once to perform clean up activities just before taking the filter from out of service.

3. doFilter()

public void doFilter(ServletRequest req,ServletResponse resp,FilterChain fc)throws SE,IOE

Total filtering logic we have to define in this method only.

This method will be executed for every request.
By using FilterChain object we can forward the request to the next level.(it may be Servlet or another Filter)

FilterConfig(I):

For every Filter web container creates a FilterConfig object and handover to the filter as argument to init() method.

By using FilterConfig object,Filter can get its configuration information.

FilterConfig defines the following methods...

1. public String getFilterName()
returns the logical name of the filter which is configured in web.xml by using <filter-name> tag.
2. public String getInitParameter(String name)
3. public Enumeration getInitParameterNames()
4. public ServletContext getServletContext()

FilterChain:

We can use FilterChain object to forward request to the next level. (It may be another Filter or Servlet)

FilterChain interface contains the following method.

public void doFilter(SR req,SR resp) throws SE,IOE

Demo Program for Filters:

DemoFilter.java:

```
1)  import javax.servlet.*;
2)  import java.io.*;
3)  public class DemoFilter implements Filter
4)  {
5)     public void init(FilterConfig conf) throws ServletException
6)     {
```

```
7)     }
8)     public void doFilter(ServletRequest req,ServletResponse resp,FilterChain fc) throws Servl
    etException,IOException
9)     {
10)       PrintWriter out = resp.getWriter();
11)       out.println("<h1>This line added by Demo Filter before processing the request</h1>")
    ;
12)       fc.doFilter(req,resp);
13)       out.println("<h1>This line added by Demo Filter after processing the request</h1>");

14)    }
15)    public void destroy()
16)    {
17)    }
18) }
```

TargetServlet.java:

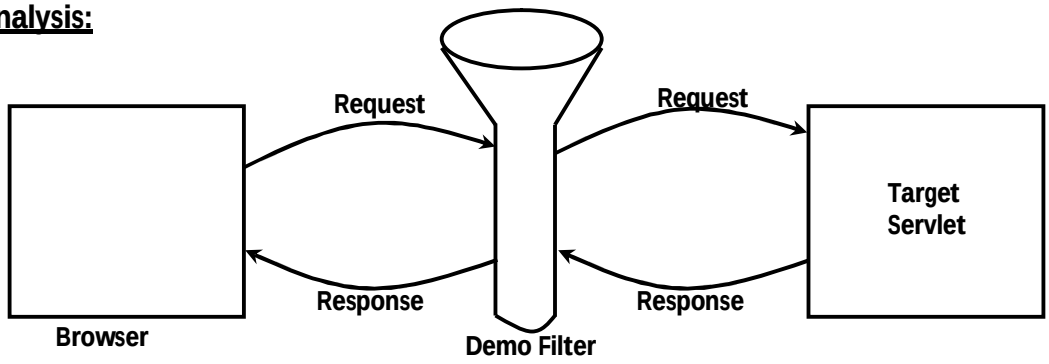
```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  public class TargetServlet extends HttpServlet
5)  {
6)    public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExce
    ption,IOException
7)     {
8)       PrintWriter out = resp.getWriter();
9)       out.println("<h1>This is the Target Servlet</h1>");
10)    }
11) }
```

web.xml:

```
1)  <web-app>
2)
3)  <servlet>
4)    <servlet-name>TargetServlet</servlet-name>
5)    <servlet-class>TargetServlet</servlet-class>
6)  </servlet>
7)
8)  <servlet-mapping>
9)    <servlet-name>TargetServlet</servlet-name>
10)   <url-pattern>/test1</url-pattern>
11) </servlet-mapping>
12)
13) <filter>
14)   <filter-name>DemoFilter</filter-name>
15)   <filter-class>DemoFilter</filter-class>
16) </filter>
```

```
17)
18) <filter-mapping>
19)   <filter-name>DemoFilter</filter-name>
20)   <url-pattern>/test1</url-pattern>
21) </filter-mapping>
22)
23) </web-app>
```

Analysis:



1. Whenever we are sending the request to TargetServlet,web container checks is there any filter configured for this servlet or not.

If any Filter configured,web container forwards the request to the Filter instead of Servlet.

After completing Filtering logic,Filter forwards the request to the TargetServlet.

After processing that request by TargetServlet , the response will be forwarded to the Filter instead of browser.

After executing Filtering logic , Filter forwards total response to the browser.

http://localhost:7777/filter1/test

Configuring Filter in web.xml:

We can configure Filter by using <filter> tag.

```
1)  <web-app>
2)    <filter>
3)       <filter-name>
4)       <filter-class>
5)       <init-param>
6)          <param-name>
7)          <param-value>
8)         </init-param>
9)    </filter>
10)   ..
11) </web-app>
```

We can map Filter either for a Particular url-pattern or to a particular Servlet or to the total web application.

Filter-Mapping to a Particular url-pattern:

```
1)  <filter-mapping>
2)    <filter-name>DemoFilter</filter-name>
3)    <url-pattern>/test1</url-pattern>
4)  </filter-mapping>
```

If the request is coming with the specified url-pattern then automatically this filter will be executed.

Filter-mapping to a Particular Servlet:

```
1)  <filter-mapping>
2)    <filter-name>DemoFilter</filter-name>
3)    <servlet-name>TargetServlet</servlet-name>
4)  </filter-mapping>
```

Filter-Mapping for Total web application:

```
1)  <filter-mapping>
2)    <filter-name>DemoFilter</filter-name>
3)    <url-pattern>*</url-pattern>
4)  </filter-mapping>
```

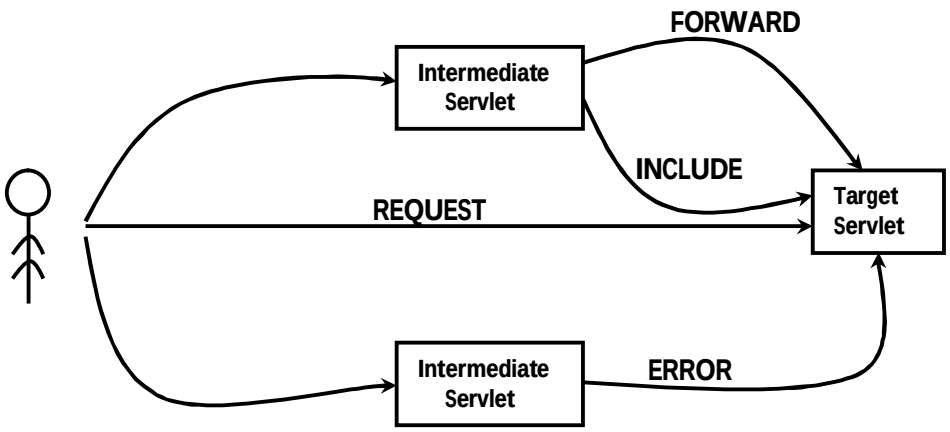
For any request to the web application whether it is for servlet or jsp, this Filter will be executed.

Note:

Mapping Filter to the total web application is possible from Servlet 2.5V onwards.

<dispatcher> tag:

A servlet can get the request in one of the following 4 ways



1. A Request Directly from Browser(REQUEST)
2. By RequestDispatcher's forward() call(FORWARD)
3. By RequestDispatcher's include() call(INCLUDE)
4. By Error Page Call(ERROR)

By default Filter concept is applicable only for direct end user REQUEST and not applicable for RD's forward,include calls and error page calls.

If we want to extend for remaining cases also then we should go for <dispatcher> tag.

<dispatcher> tag introduced in Servlet 2.4V.

The allowed values for <dispatcher> tag are:

1. REQUEST:
Filter will be executed for direct end user request.
This is default value

2. FORWARD:
Filter will be executed for RD's forward() call.

3. INCLUDE:
Filter will be executed for RD's include() call

4. ERROR:
Filter will be executed for Error page call

```
1)    <error-page>
2)      <exception-type>java.lang.AE</exception-type>
3)      <location>/test</location>
4)    </error-page>
```

case-1:
If we want to execute Filter for direct end user's request and RD's forward() call, then we have to configured filter as follows...

```
1)  <filter-mapping>
2)    <filter-name>DemoFilter</filter-name>
3)    <url-pattern>/test1</url-pattern>
4)    <dispatcher>REQUEST</dispatcher>
5)    <dispatcher>FORWARD</dispatcher>
6)  </filter-mapping>
```

In this case Filter won't be executed for RD's include call and error page calls.

Case-2:

```
1)  <filter-mapping>
```

```
2)    ....
3)    <dispatcher>INCLUDE</dispatcher>
4)  </filter-mapping>
```

In this case Filter will be executed only for RD's include call and in the remaining 3 cases filter wont be executed.

Demo Program for <dispatcher> tag:

web.xml:

```
1)  <web-app>
2)
3)  <servlet>
4)    <servlet-name>FirstServlet</servlet-name>
5)    <servlet-class>FirstServlet</servlet-class>
6)  </servlet>
7)
8)  <servlet>
9)    <servlet-name>TargetServlet</servlet-name>
10)   <servlet-class>TargetServlet</servlet-class>
11) </servlet>
12) <servlet>
13)   <servlet-name>TargetServlet1</servlet-name>
14)   <servlet-class>TargetServlet1</servlet-class>
15) </servlet>
16)
17) <servlet-mapping>
18)   <servlet-name>FirstServlet</servlet-name>
19)   <url-pattern>/test1</url-pattern>
20) </servlet-mapping>
21)
22) <servlet-mapping>
23)   <servlet-name>TargetServlet</servlet-name>
24)   <url-pattern>/test2</url-pattern>
25) </servlet-mapping>
26) <servlet-mapping>
27)   <servlet-name>TargetServlet1</servlet-name>
28)   <url-pattern>/test3</url-pattern>
29) </servlet-mapping>
30) <filter>
31)   <filter-name>DemoFilter</filter-name>
32)   <filter-class>DemoFilter</filter-class>
33) </filter>
34)
35) <filter-mapping>
36)   <filter-name>DemoFilter</filter-name>
37)   <servlet-name>TargetServlet</servlet-name>
38)   <dispatcher>REQUEST</dispatcher>
```

```
39)  <dispatcher>FORWARD</dispatcher>
40)  <dispatcher>INCLUDE</dispatcher>
41)  <dispatcher>ERROR</dispatcher>
42) </filter-mapping>
43)
44) <error-page>
45)   <exception-type>java.lang.ArithmeticException</exception-type>
46)   <location>/test2</location>
47)  </error-page>
48) </web-app>
```

TargetServlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  public class TargetServlet extends HttpServlet
5)  {
6)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
7)     {
8)        PrintWriter out = resp.getWriter();
9)        out.println("<h1>This is the Target Servlet</h1>");
10)    }
11) }
```

TargetServlet1.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  public class TargetServlet1 extends HttpServlet
5)  {
6)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
7)     {
8)        PrintWriter out = resp.getWriter();
9)        out.println(10/0);
10)    }
11) }
```

FirstServlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  public class FirstServlet extends HttpServlet
5)  {
```

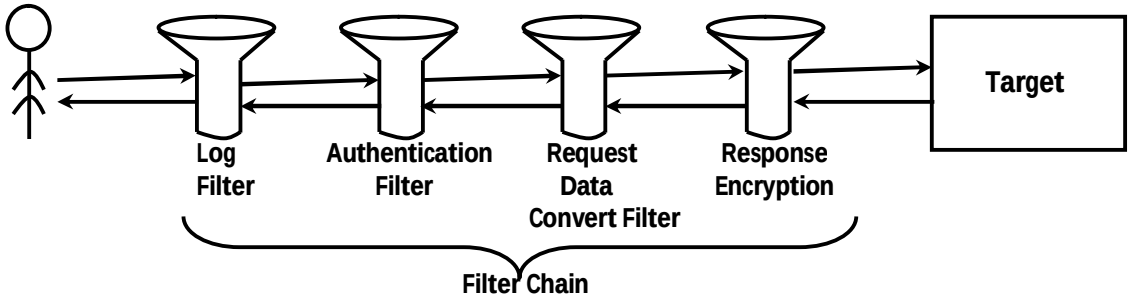
```
6)    public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
      eption,IOException
7)    {
8)       RequestDispatcher rd = req.getRequestDispatcher("/test2");
9)       rd.include(req,resp);
10)      //rd.forward(req,resp);
11)   }
12) }
```

DemoFilter.java:

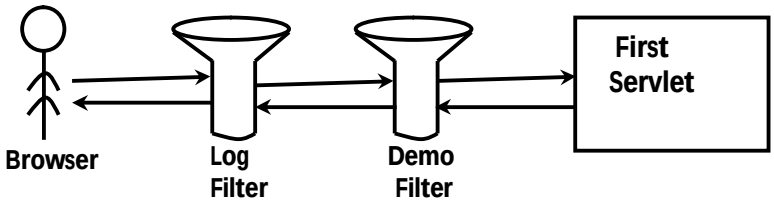
```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  public class DemoFilter implements Filter
5)  {
6)    public void init(FilterConfig conf) throws ServletException
7)    {
8)    }
9)    public void doFilter(ServletRequest req,ServletResponse resp,FilterChain fc) throws Servl
      etException,IOException
10)   {
11)      PrintWriter out = resp.getWriter();
12)      out.println("<h1>This line added by Demo Filter before processing the request</h1>")
      ;
13)      fc.doFilter(req,resp);
14)      out.println("<h1>This line added by Demo Filter after processing the request</h1>");

15)   }
16)   public void destroy()
17)   {
18)   }
19) }
```

Filter Chain:

We can configure more than one Filter for a Target Resource and all these filters will be executed one by one and forms Filter Chain.



Demo Program for FilterChain:



web.xml:

```
1)  <web-app>
2)
3)  <filter>
4)    <filter-name>LogFilter</filter-name>
5)    <filter-class>LogFilter</filter-class>
6)  </filter>
7)
8)  <filter>
9)    <filter-name>DemoFilter</filter-name>
10)   <filter-class>DemoFilter</filter-class>
11) </filter>
12)
13)  <filter-mapping>
14)   <filter-name>LogFilter</filter-name>
15)   <url-pattern>/test1</url-pattern>
16)  </filter-mapping>
17)
18)   <filter-mapping>
19)   <filter-name>DemoFilter</filter-name>
20)   <url-pattern>/test1</url-pattern>
21)  </filter-mapping>
22)
23) <servlet>
24)   <servlet-name>FirstServlet</servlet-name>
25)   <servlet-class>FirstServlet</servlet-class>
26) </servlet>
27)
28) <servlet-mapping>
29)   <servlet-name>FirstServlet</servlet-name>
30)   <url-pattern>/test1</url-pattern>
31) </servlet-mapping>
32)
33) </web-app>
```

FirstServlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
```

```
3)  import java.io.*;
4)  public class FirstServlet extends HttpServlet
5)  {
6)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
7)     {
8)        PrintWriter out = resp.getWriter();
9)        out.println("<h1>This is Target Servlet</h1>");
10)    }
11) }
```

DemoFilter.java:

```
1)  import javax.servlet.*;
2)  import java.io.*;
3)  public class DemoFilter implements Filter
4)  {
5)     public void init(FilterConfig conf) throws ServletException
6)     {
7)     }
8)     public void doFilter(ServletRequest req,ServletResponse resp,FilterChain fc) throws Servl
    etException,IOException
9)     {
10)       PrintWriter out = resp.getWriter();
11)       out.println("<h1> This Line added by DemoFilter before processing of the request</h1
    >");
12)       fc.doFilter(req,resp);
13)       out.println("<h1> This Line added by DemoFilter After processing of the request</h1>
    ");
14)    }
15)    public void destroy()
16)    {
17)    }
18) }
```

LogFilter.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.util.*;
4)  import java.io.*;
5)  public class LogFilter implements Filter
6)  {
7)     private FilterConfig config;
8)     public void init(FilterConfig config) throws ServletException
9)     {
10)       this.config= config;
11)    }
```

```
12)   public void doFilter(ServletRequest req,ServletResponse resp,FilterChain fc) throws Servl
    etException,IOException
13)   {
14)      PrintWriter out = resp.getWriter();
15)      out.println("<h1> This Line added by log Filter before processing of the request</h1>"
    );
16)      ServletContext context = config.getServletContext();
17)      HttpServletRequest req1 = (HttpServletRequest)req;
18)      context.log("A request is coming from "+req1.getRemoteHost()+" for URL :"+req1.get
    RequestURL()+" at "+ new Date());
19)      fc.doFilter(req,resp);
20)      out.println("<h1> This Line added by Log Filter After processing of the request</h1>");

21)   }
22)   public void destroy()
23)   {
24)      config= null;
25)   }
26) }
```

Note:

In filter3 demo program, the log file is available in the following location:

Tomcat/logs/localhost.2017-02-26.txt

Web Container's Rule for ordering of Filters in FilterChain:

1. Identify all filters which are configured according to url-pattern & execute all these filters from top to bottom.

2. Identify all filters which are configured according to <servlet-name> & execute all these filters from top to bottom.

Note:

<filter> and <filter-mapping> tags are direct child tags of <web-app> and hence we can take these tags anywhere within <web-app>.

Difference b/w Filter's doFilter() Method and FilterChain's doFilter() Method:

Filter's doFilter() Method	FilterChain's doFilter() Method
1) public void doFilter (SR req, SR resp, FC fc) throws SE, IOE	1) public void doFilter (SR req, SR resp) throws SE, IOE
2) Total Filtering Logic, we have to define in this Method only.	2) To forward Request to the next Level (It may be another Filter OR Servlet)
3) It is call back method because Web Container calls this Method automatically.	3) It is not call back method because we have to call explicitly this Method.

Wrappers

Sometimes it is required to alter request and response information in the filters. We can implement this by using wrapper classes.

i.e we can use wrappers inside filters to alter request and response information.

Eg1:
Within the filter,we have to convert end user's resume from word format to pdf format.

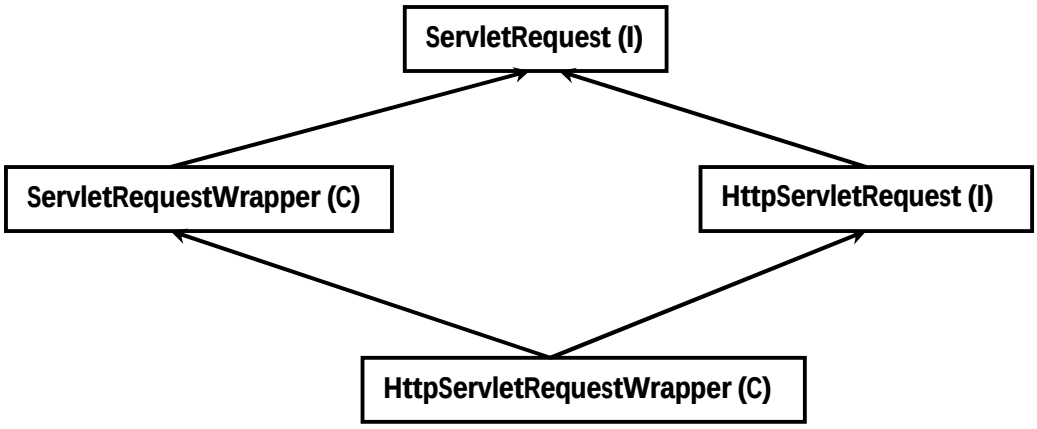
Eg 2:
Within the filter,we have to compress the response and that compressed response we can send to the browser,so that we can reduce download time.

There are 2 types of wrappers

1. Request wrappers
2. Response wrappers

1. Request wrappers:

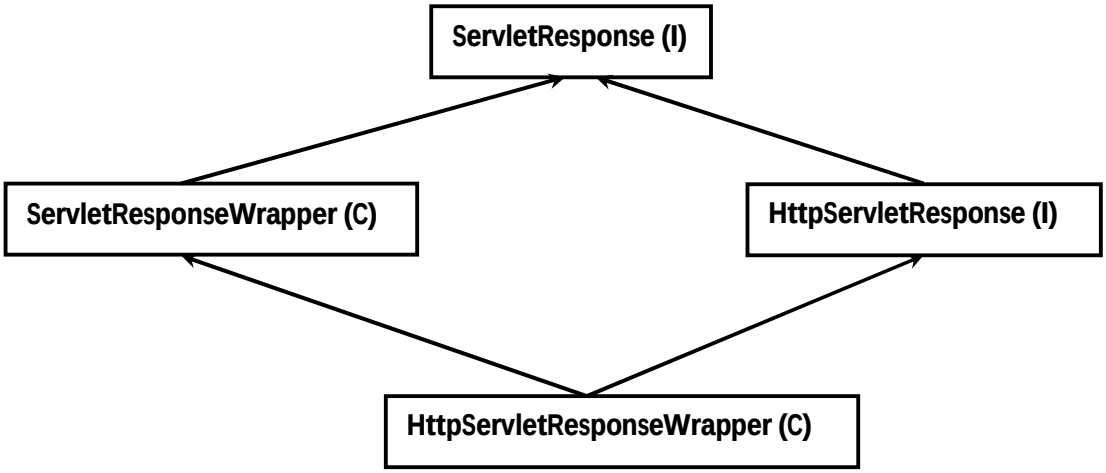
To alter request information
There are two request wrapper classes...



1. ServletRequestWrapper
2. HttpServletRequestWrapper

2. Response wrappers:

To alter response information



There are two response wrapper classes
1. ServletResponseWrapper
2. HttpServletResponseWrapper

Demo Program for Request Wrapper:

login.html:

```
1)  <html>
2)  <body bgcolor=green text=white><center><h1>Durga Software Solutions</h1></center>
3)  <form action = "/wrapper/test1" >
4)  <h1>Enter Any Word :<input type=text name=word></h1>
5)  <input type=submit>
6)  </form>
7)  </body>
8)  </html>
```

CustomizedRequest.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  public class CustomizedRequest extends HttpServletRequestWrapper
4)  {
5)     public CustomizedRequest(HttpServletRequest req)
6)     {
7)        super(req);// to make original request information available to the Parent class
8)     }
9)    public String getParameter(String word)
10)    {
11)      String word1 = super.getParameter(word);
12)      if(word1.equals("JAVA") | word1.equals("SCJP") | word1.equals("SCWCD") | word1.e
     quals("SAT"))
13)         return "SLEEP";
```

```
14)      else
15)      return word1;
16)   }
17)  }
```

BadWordFilter.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  public class BadWordFilter implements Filter
5)  {
6)     public void init(FilterConfig conf) throws ServletException
7)     {
8)     }
9)     public void doFilter(ServletRequest req,ServletResponse resp,FilterChain fc) throws Servl
    etException,IOException
10)    {
11)       CustomizedRequest req1 = new CustomizedRequest((HttpServletRequest)req);
12)       fc.doFilter(req1,resp);
13)    }
14)    public void destroy()
15)    {
16)    }
17) }
```

TargetServlet.java:

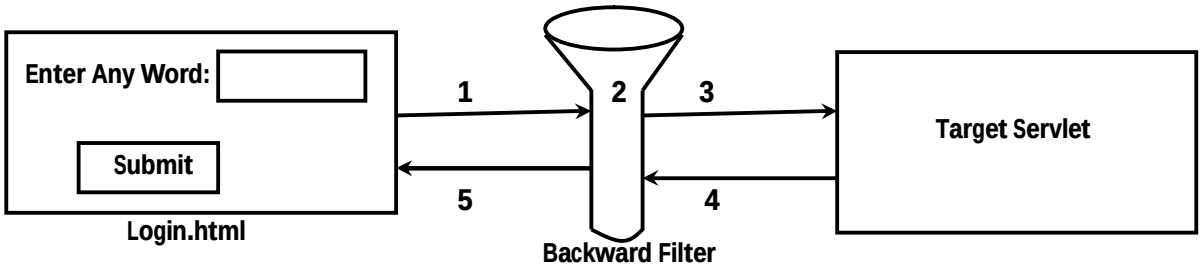
```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  public class TargetServlet extends HttpServlet
5)  {
6)
7)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
8)     {
9)        PrintWriter out = resp.getWriter();
10)       String word = req.getParameter("word");
11)       out.println("<h1>Hi your typed word is :"+word+"</h1>");
12)    }
13) }
```

web.xml:

```
1)  <web-app>
2)  <filter>
3)    <filter-name>BadWordFilter</filter-name>
4)    <filter-class>BadWordFilter</filter-class>
```

```
5)  </filter>
6)  <filter-mapping>
7)    <filter-name>BadWordFilter</filter-name>
8)    <servlet-name>TargetServlet</servlet-name>
9)  </filter-mapping>
10)
11) <servlet>
12)   <servlet-name>TargetServlet</servlet-name>
13)   <servlet-class>TargetServlet</servlet-class>
14) </servlet>
15)
16) <servlet-mapping>
17)   <servlet-name>TargetServlet</servlet-name>
18)   <url-pattern>/test1</url-pattern>
19) </servlet-mapping>
20) </web-app>
```

Flow of Program-Execution:



End user entered the word and click the submit button.
Filter creates a CustomizedRequest object by using wrapper class.
Filter forwards that customized request object to the TargetServlet instead of original request.
Within the servlet if we are performing any operation o the request object,our own customized behaviour will be reflected.
TargetServlet prepares the response and forwards to Filter.
Filter forwards that response inturn to the end user.

Note:
We are not required to configure anything related to wrapper in web.xml.

Conclusions:

1. Filter object will be created by web container automatically. For this web container always calls public no-arg constructor.Hence every Filter class should compulsary contains public no-arg constructor. It may be explicitly provided by programmer or default constructor generated by compiler.

2. Filter object will be created automatically by the web container at the time of application deployment or at the time of server startup. Hence <load-on-startup> is not required for the filters.

3. Usage of filter is nothing but following
Intercepting Filter Design Pattern.Hence it is recommended to use Filters concept in our application.

4. Usage of Wrappers is nothing but following Decorator design pattern. Hence it is recommended to use Wrappers in our application.

<u>@WebFilter Annotation in Servlet 3.0V:</u>
It is the replacement for filter configurations in web.xml

```
import javax.servlet.annotation.*;
@WebFilter(filterName="DemoFilter", urlPatterns="/test")
public class DemoFilter implements Filter
{

 ....
}
```

Unit 4: Session Management

Objective:

1. For the given scenario, describe the session API?
2. Explain the process of creating a Session Object?
3.What are various different mechanisms to invalidate a session?

Client and Server can communicates with some common language, which is nothing but HTTP.

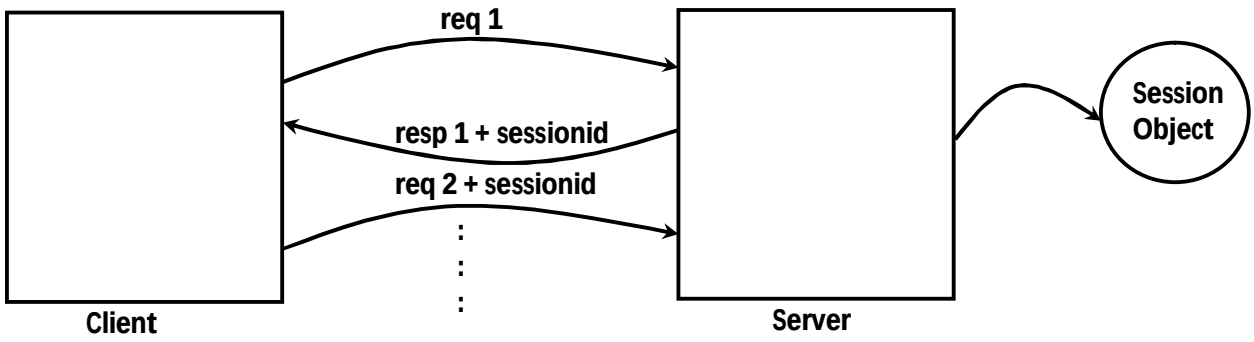
The basic limitation of HTTP is, it is stateless protocol. i.e it is unable to remember client information for future purpose across multiple requests.Every request to the server is treated as a new request.

Hence some mechanism is required at server side to remember client information across multiple requests. This mechanism is nothing but session management mechanism.

The following are various session management mechanisms.

1. Session API
2. Cookies
3. URL Rewriting
4. Hidden Form Fields [It is not official mechanism from SUN,it is just programmer's trick to remember client information]

Session Management by using Session API:



Whenever client sends a request to the server, if server wants to remember client information for the future purpose then server will create a session object and stores the required information in the form of attributes.Server sends the corresponding session id to the browser as the part of response.

With every consecutive request,browser sends that session id. By accessing session id and the corresponding session object,server can able to remember client info across multiple requests.

Client information will be maintained at server side in session object in the form of attributes.

Process of Creating Session Object:

HttpServletRequest interface defines the following methods for creating session object.

1. public HttpSession getSession()

Eg: HttpSession session = req.getSession();

First this method will check is there any session aleady associated with request object or not.

If the request does not associated with any session, then this method creates a new session object and returns it.

If the request already associated with session object then existing session object will be returned. There is a guarentee that this method will always return session object.It may be newly created or already existing one.

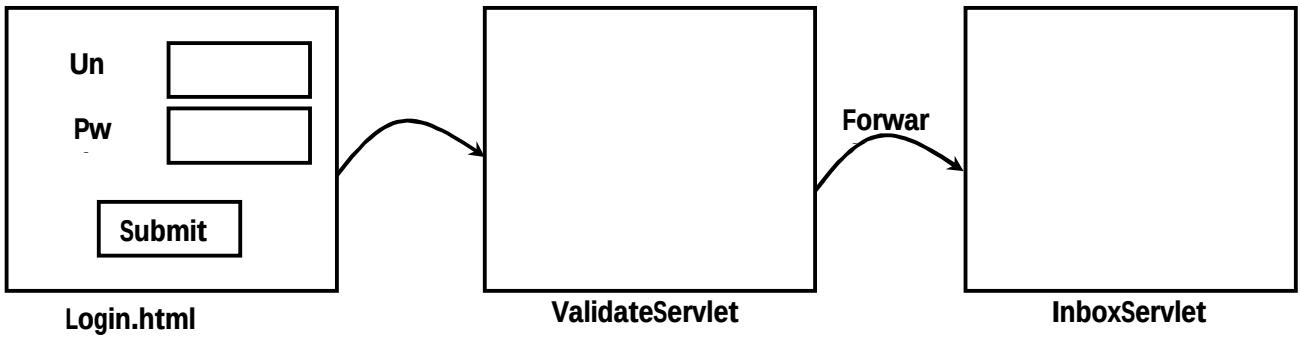
2. public HttpSession getSession(boolean b)

If the argument is true then this method simply acts as getSession().

If the argument is false, then this method first checks whether the request associated with any session or not.If the request already associated with session then this method returns existing session object.

If the request does not associated with any session then this method returns null without creating any new session object.

Case Study:



ValidateServlet will check whether credentials are valid or not. If valid then it is responsible to create session object. Hence inside ValidateServlet we have to use getSession() method.

HttpSession session =req.getSession();

After creating session object ValidateServlet forwards the request to InboxServlet.

To Access InboxServlet compulsary the request should be associated with session. If the request does not associated with any session, then it is not responsible to create session object and it will forward the request to login page. Hence in this case we have to use getSession(false)

```
1)  HttpSession session =req.getSession(false);
2)  if(session==null)
3)  {
4)     forward request to login page
5)  }
6)  else
7)  {
8)     display inbox page
9)  }
```

Q. Which of the following are equal?

1. session = req.getSession();
2. session = req.getSession(false);
3. session = req.getSession(true);
4. session = resp.getSession();

Ans:1 and 3

Invalidating session object:

We can invalidate a session by using the following 2 ways
1. By using invalidate() method
2. By Timeout mechanism

invalidate() method:

HttpSession interface defines invalidate() method to invalidate session explicitly.
public void invalidate()

whenever we are clicking logout button internally this method will be executed.

session.invalidate()

2. Timeout mechanism:

If we are not performing any operation on the session object for a pre defined amount of time then the session will be expired automatically. This predefined amount of time is called session timeout.

We can configure session timeout either server level or web application level or a particular session object level.

1. Session Timeout at Server Level:

Most of the web servers provide default support for session timeout. Mostly it is 30 minutes.
We are allowed to change this server level session timeout based on our requirement.
This session timeout applicable for all sessions created in that server of all web applications.

2. Configuring session timeout at web application level:

If we are not satisfied with server level session timeout then we have to configure at application level.
We can configure session time out at application level in web.xml as follows...

```
1)  <web-app>
2)    ...
3)    <session-config>
4)       <session-timeout>10</session-timeout>
5)    </session-config>
6)  </web-app>
```

<session-config> is the child tag of <web-app> and hence we can place anywhere within <web-app>
The unit to the <session-timeout> is minutes
zero or -ve value indicates that session never expires.
This session timeout is applicable for all the session s which are created in that web application.

3. Setting session timeout for a particular session object:

We can set session timeout for a particular session object by using the following method of HttpSession.
public void setMaxInactiveInterval(int seconds)
The argument is in seconds
-ve value indicates that session never expires
zero value indicates that session will expire immediately.
This session timeout is applicable only for a particular session object on which we call this method.

Comparison between 2 Session Timeout Mechanisms:

Property	<session-timeout>	setMaxInactiveInterval()
1) Scope	It is applicable for all Sessions which are created in that Web Application	It is applicable only for a particular Session Object, on which we called this Method
2) Units	Minutes	Seconds
3) 0 Value	Indicates that Session never expires	Sessions will expire immediately
4) -ve Value	Indicates that Session never expires	Indicates that Session never expires

Q. How we can implement Log out mechanism?

2 ways.

1st way:

```
session.invalidate();
```

2nd way:

```
session.setMaxInactiveInterval(0);
```

```
public class LogOutServlet extends HttpServlet
{
  doGet(..)...
  {
    HttpSession session = req.getSession(false);
    if(session != null)
    {
      session.invalidate();
    }
  }
}
```

Note:

If we configured session timeout in all 3 ways then timeout at particular session object will be considered.

Important Methods of HttpSession:

1. public boolean isNew()

To check whether the session object is newly created or not

2. public void invalidate()

to expire a session forcefully

3. public void setMaxInactiveInterval(int seconds)

To set session timeout for a particular session object

4. public int getMaxInactiveInterval()

Returns the session timeout value in seconds

5. public String getId()

Returns session id

6. public long getCreationTime()

Returns the time when the session was created in milli seconds since Jan 1st 1970.

If we are passing this long value to the Date constructor then we will get exact Date and time.

Eg:

```
long ms = session.getCreationTime();
Date d = new Date(ms);
SOP(d);
```

7. public long getLastAccessedTime()

Returns the time when the client accessed session recently in milli seconds since 1970 Jan 1st.

8. public ServletContext getServletContext()

Returns the ServletContext object to which this session belongs

HttpSession interface defines the following methods to perform attribute management in session scope.

1. public void setAttribute(String name,Object value)
2. public Object getAttribute(String name)
3. public void removeAttribute(String name)
4. public Enumeration getAttributeNames()

Note: Once session expired,we are not allowed to call most of above methods.Otherwise we will get RE saying IllegalStateException

Demo Program for session management by using Session API:

login.html:

```
1)  <html>
2)  <body>
3)  <form action="/session1/test1">
4)  <h1>Enter Books Information</h1>
5)  <pre>
6)  <h2>Name:<input type="text" name="name">
7)  Value:<input type="text" name="value">
8)  </h2></pre>
9)
10) <input type="submit" value="Add To Cart" />
11) </form>
12) <a href="/session1/test2">Show My Cart</a>
13) </body>
14) </html>
```

SessionServlet1.java:

```
1)  import java.io.*;
2)  import javax.servlet.*;
3)  import javax.servlet.http.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test1")
6)  public class  SessionServlet1 extends HttpServlet
```

```
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse resp)throws ServletExce
    ption,IOException
9)     {
10)       PrintWriter out=resp.getWriter();
11)       HttpSession hs=req.getSession();
12)       if(hs.isNew())
13)       {
14)          out.println("<h2>New Session got created with session ID:"+hs.getId()+"</h2>");
15)       }
16)       else
17)       {
18)          out.println("<h2>Existing  Session only using  with session ID:"+hs.getId()+"</h2>");
19)       }
20)       String name=req.getParameter("name");
21)       String value=req.getParameter("value");
22)       hs.setAttribute(name,value);
23)       //hs.setMaxInactiveInterval(120);
24)       RequestDispatcher rd = req.getRequestDispatcher("login.html");
25)       rd.include(req,resp);
26)    }
27) }
```

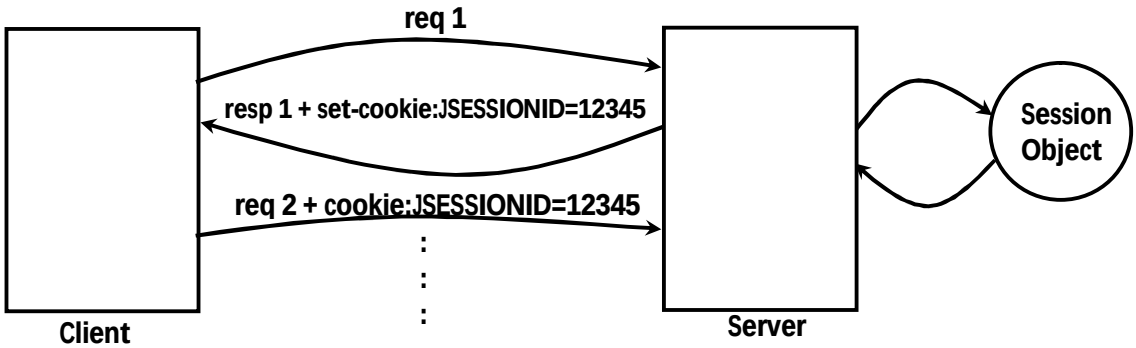
SessionServlet2.java:

```
1)  import java.io.*;
2)  import javax.servlet.*;
3)  import javax.servlet.http.*;
4)  import java.util.*;
5)  import javax.servlet.annotation.*;
6)  @WebServlet("/test1")
7)  public class SessionServlet2 extends HttpServlet
8)  {
9)     public void doGet(HttpServletRequest req,HttpServletResponse res)throws ServletExcep
    tion,IOException
10)    {
11)       PrintWriter out=res.getWriter();
12)       HttpSession hs=req.getSession(false);
13)       if(hs == null)
14)       {
15)          out.println("<h2> No session information is available</h2>");
16)       }
17)       else
18)       {
19)       Enumeration e = hs.getAttributeNames();
20)       out.println("<table border=2><tr><th>Session AttributeName</th><th>Sesion Attribu
    te value</th></tr>");
21)       while(e.hasMoreElements())
22)       {
```

```
23)         String name = (String)e.nextElement();
24)         String value = (String)hs.getAttribute(name);
25)         out.println("<tr><td>"+name+"</td><td>"+value+"</td></tr>");
26)       }
27)       out.println("</table>");
28)       long l1 = hs.getCreationTime();
29)       long l2 = hs.getLastAccessedTime();
30)       int l3 = hs.getMaxInactiveInterval();
31)       out.println("<h3>The creation time is "+new Date(l1)+"</h3>");
32)       out.println("<h3>The last accessed time is "+new Date(l2)+"</h3>");
33)       out.println("<h3>Max inactive interval is :"+l3+"</h3>");
34)       }
35)    }
36) }
```

session1
  |-login.html
  |-WEB-INF
    |-classes
      |-SessionServlet1.class
      |-SessionServlet2.class

How the session id exchanging b/w Client and Server:



Whenever browser sends a request to server,If server wants to remember client information for the future purpose,then Server will create Session object and store required information in the form of attributes.Server sends the corresponding sessionid as the part of response.For this server will use setCookie response header.

Browser will retrieve that session id and will send with every consecutive request to the server.For this browser will use cookie request header.

Hence session id exchanging b/w client and server with setCookie response header and cookie request header.

Demo Program to demonstrate how session id is exchanging b/w client and server:

login.html:

```
<html>
<body>
<form action="/session2/test1">
<h1>Enter session information</h1>
<pre>
<h2>Name:<input type="text" name="uname">
Value:<input type="text" name="uvalue">
</h2></pre>

<input type="submit"/>
</form>
<a href="/session2/test2">RequestHeader Information</a>
</body>
</html>
```

SessionServlet1.java:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;
@WebServlet("/test1")
public class  SessionServlet1 extends HttpServlet
{
   public void doGet(HttpServletRequest req,HttpServletResponse res)throws ServletException,IOException
   {
      PrintWriter out=res.getWriter();
      HttpSession hs=req.getSession();
      if(hs.isNew())
      {
         out.println("<h2>New Session got created with session ID:"+hs.getId()+"</h2>");
      }
      else
      {
         out.println("<h2>Existing  Session only using  with session ID:"+hs.getId()+"</h2>");
      }
      String name=req.getParameter("uname");
      String value=req.getParameter("uvalue");
      hs.setAttribute(name,value);
      //hs.setMaxInactiveInterval(120);
      RequestDispatcher rd = req.getRequestDispatcher("login.html");
      rd.include(req,res);
   }
}
```

RequestHeaderDemoServlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  import java.util.*;
5)  import javax.servlet.annotation.*;
6)  @WebServlet("/test2")
7)  public class RequestHeaderDemoServlet extends HttpServlet
8)  {
9)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
10)   {
11)      PrintWriter out = resp.getWriter();
12)      out.println("<h1>Request Headers Information</h1></hr>");
13)      out.println("<table border=2><tr><th>HeaderName</th><th>Header values</th></tr
    >");
14)      Enumeration e = req.getHeaderNames();
15)      while(e.hasMoreElements())
16)      {
17)         String hname = (String)e.nextElement();
18)         out.println("<tr><td>"+hname+"</td><td>"+ req.getHeader(hname)+"</td></tr>");
19)      }
20)      out.println("</table></body></html>");
21)   }
22) }
```

```
session2
  |-login.html
  |-WEB-INF
    |-classes
       |-SessionServlet1.class
       |-RequestHeaderDemoServlet.class
```

Note:

If the required session information is very less then creating a seperate session object and maintaining that object at server side is not recommended b'z it creates performance problems.

To resolve this,we should go for Cookies concept,where session information is maintained at client side & server is not responsible to maintain session info.

Session Management by using Cookies:

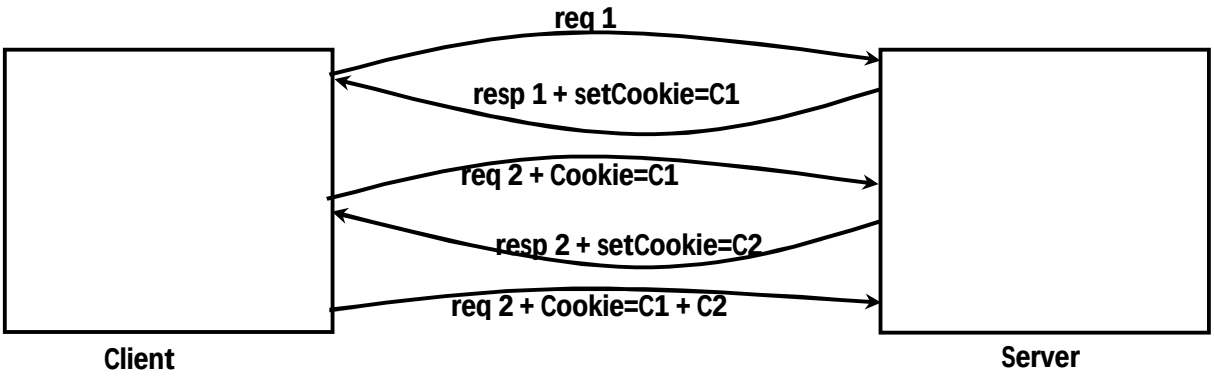
Cookie is a small amount of information(key-value pair),which is created by server and maintained by client.

Whenever browser sends first request,if server wants to remember client information for the future purpose, then server will creates Cookie object with the required information and sends to the browser as the part of response.

Browser stores that cookie in the local file system and sends to the server with every consecutive request.By accessing that cookie server can able to remember client information.

Server will use setCookie response header to send cookies to the client. Browser will use cookie request header to send cookies to the server.

Hence by using setCookie response header and cookie request header cookies are exchanging b/w client and server. It is exactly same as exchanging sessionid b/w client and server.



We can create a Cookie object by using Cookie class Constructor.

Cookie c = new Cookie(String name,String value);

Eg:
Cookie c = new Cookie("durga","10");

After creating Cookie object,we have to add that object to the response by using addCookie() method.

resp.addCookie(c);

At server side we can retrieve cookies send by the client from request object by using getCookies() method.

Cookie[] c = req.getCookies();

If the request does not associated with any cookies then this method returns null.

Important methods of Cookie class:

1. public String getName()
   returns the name of the Cookie

2. public String getValue()
   returns the value of the Cookie

3. public int getMaxAge()
   Returns the max age of the Cookie in seconds.

4. public void setMaxAge(int seconds)

To set max age of the cookie.

setting max age as -1,then cookies will be expired automatically whenever browser window closed.

-1 is the default value.

Demo Program for session Management by using Cookies:

cookie.html:

```
1)  <html>
2)  <body>
3)  <form action="/session3/test1">
4)  <h1>Enter cookie information</h1>
5)  <pre>
6)  Name:<input type="text" name="uname">
7)  Value:<input type="text" name="uvalue">
8)  </pre>
9)  <input type="submit"/>
10) </form>
11) <a href="/session3/test2">View Cookies</a>
12) </body>
13) </html>
```

CookieDemoServlet1.java:

```
1)  import java.io.*;
2)  import javax.servlet.*;
3)  import javax.servlet.http.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test1")
6)  public class  CookieDemoServlet1 extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse res)throws ServletExcep
    tion,IOException
9)     {
10)       PrintWriter out=res.getWriter();
11)       String name=req.getParameter("uname");
12)       String value=req.getParameter("uvalue");
13)       Cookie c = new Cookie(name,value);
14)       c.setMaxAge(180);
15)       res.addCookie(c);
16)       out.println("<h2>Cookie added successfully</h2>");
17)       RequestDispatcher rd = req.getRequestDispatcher("cookie.html");
18)       rd.include(req,res);
19)    }
20) }
```

CookieDemoServlet2.java:

```
1)  import java.io.*;
2)  import javax.servlet.*;
3)  import javax.servlet.http.*;
4)  import java.util.*;
5)  import javax.servlet.annotation.*;
6)  @WebServlet("/test2")
7)  public class CookieDemoServlet2 extends HttpServlet
8)  {
9)     public void doGet(HttpServletRequest req,HttpServletResponse res)throws ServletExcep
    tion,IOException
10)   {
11)      PrintWriter out=res.getWriter();
12)      Cookie[] c = req.getCookies();
13)      if(c == null)
14)      {
15)         out.println("<h2> No cookies are assiciated with the request</h2>");
16)      }
17)      else
18)      {
19)       out.println("<table border=2><tr><th>Cookie Name</th><th>Cookie Value</th></tr>
    ");
20)      for(Cookie c1: c)
21)      {
22)         String name = c1.getName();
23)         String value = c1.getValue();
24)         out.println("<tr><td>"+name+"</td><td>"+value+"</td></tr>");
25)      }
26)      out.println("</table>");
27)       }
28)   }
29) }
```

```
session3
  |-cookie.html
  |-WEB-INF
    |-classes
       |-CookieDemoServlet1.class
       |-CookieDemoServlet2.class
```

Demo Program how cookies are exchanging b/w client and server:

cookie.html:

```
1)  <html>
2)  <body>
3)  <form action="/session4/test1">
4)  <h1>Enter cookie information</h1>
```

```
<pre>
Name:<input type="text" name="uname">
Value:<input type="text" name="uvalue">
</pre>
<input type="submit"/>
</form>
<a href="/session4/test2">View Request Headers</a>
</body>
</html>
```

CookieDemoServlet1.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;
@WebServlet("/test1")
public class  CookieDemoServlet1 extends HttpServlet
{
   public void doGet(HttpServletRequest req,HttpServletResponse res)throws ServletException,IOException
   {
      PrintWriter out=res.getWriter();
      String name=req.getParameter("uname");
      String value=req.getParameter("uvalue");
      Cookie c = new Cookie(name,value);
      //c.setMaxAge(120);
      res.addCookie(c);
      out.println("<h2>Cookie added successfully</h2>");
      RequestDispatcher rd = req.getRequestDispatcher("cookie.html");
      rd.include(req,res);
   }
}
```

RequestHeaderDemoServlet.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import javax.servlet.annotation.*;
@WebServlet("/test2")
public class RequestHeaderDemoServlet extends HttpServlet
{
   public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletException,IOException
   {
      PrintWriter out = resp.getWriter();
      out.println("<h1>Request Headers Information</h1></hr>");
```

```
13)      out.println("<table border=2><tr><th>HeaderName</th><th>Header values</th></tr
    >");
14)      Enumeration e = req.getHeaderNames();
15)      while(e.hasMoreElements())
16)      {
17)         String hname = (String)e.nextElement();
18)         out.println("<tr><td>"+hname+"</td><td>"+ req.getHeader(hname)+"</td></tr>");

19)      }
20)      out.println("</table></body></html>");
21)   }
22) }
```

session4
  |-cookie.html
  |-WEB-INF
    |-classes
      |-CookieDemoServlet1.class
      |-RequestHeaderDemoServlet.class

Persistant cookies vs non-persistant cookies:

If we are setting max age to the cookie,then such type of cookies are called persistant cookies or permanent cookies. These will be stored in the local file system of the client.

If we are not setting max age ,then such type of cookies are called temporary cookies or non-persistant cookies.These cookies will be stored in the browser's cache and not visible in the local file system. Once we close the browser, automatically these cookies will be expired.

Advantages of Cookies:
1. Very easy to implement
2. Persist across server restarts also
3. All browsers and servers provide automatic support for cookies.

Disadvantages of Cookies:
1. Cookies can be enabled or disabled at client side to meet security constraints.
If the cookies are disabled then session management by using cookies is not possible.
2.The number of cookies supported by any browser is always fixed.
3. The max size of the cookie is also fixed. Hence we can not store huge amount of information by using Cookies.
4. Cookie data is always String type.

Differences b/w Session API and Cookies:

Session API	Cookies
1) Session Information will be maintained at Server side.	1) Session Information will be maintained at Client side.
2) Best suitable if we want to store huge amount of Information.	2) Best suitable if we want to store less amount of Information.
3) Session Information need not be String Type.	3) Session Information should be String Type.
4) Network Problems won't be raised.	4) There may be a chance of Network Problems.
5) Security is more.	5) Security is less.

If Cookies are disabled at Client Side then what will happend?

If the cookies are disabled at client side then browser is unable to see Set-Cookie response header. Hence browser wont get any cookies or session id send by server.

If the cookies are disabled at client side then browser unable to send Cookie request header.Hence server wont get any cookies or session id from the request and every request is treated as new request. Due to this total session management fails.

To overcome this problem,we should go for the most powerful and painful technique: URL REWRITING.

Session Management by URL REWRITING

URLs can be re written or encoded to include session information.This technique is called url rewriting.

URL Rewriting=URL+Session Info

Eg: url;JSESSIONID=1234

HttpServletResponse defines the following methods to append session id to the url.

1. public String encodeURL(String url)
   Returns url by appending JSESSIONID.

2. public String encodeRedirectURL(String url)
   Returns url by appending session id.
   This can be used as argument to sendRedirect() method.

The above 2 methods will append JSESSIONID to the url iff cookies are disabled at client side.

If the cookies are enabled,these methods return the same url without appending JSESSIONID.

At server side we can identify whether sessionid is coming as the part of url or from the Cookie request header by using the following methods of HttpServletRequest.

1. public boolean isRequestedSessionIdfromURL()
2. public boolean isRequestedSessionIdfromCookie()

By using these methods we can identify underlying session management technique.

Demo Program for session management by url rewriting:

login.html:

```
1)  <html>
2)  <body>
3)  <form action="/session5/test1">
4)  <h1>Enter session information<br>
5)  Name:<input type="text" name="uname"><br>
6)  <input type="submit"/></h1>
7)  </form>
8)  </body>
9)  </html>
```

SessionServlet1.java:

```
1)  import java.io.*;
2)  import javax.servlet.*;
3)  import javax.servlet.http.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test1")
6)  public class  SessionServlet1 extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse res)throws ServletExcep
    tion,IOException
9)     {
10)       PrintWriter out=res.getWriter();
11)       String name=req.getParameter("uname");
12)       out.println("<h1>Welcome to Durga Software Solutions</h1>");
13)       out.println("<a href=/session5/test2?name="+name+">Click here to get user name</a
    >");
14)    }
15) }
```

SessionServlet2.java:

```
1)  import java.io.*;
2)  import javax.servlet.*;
3)  import javax.servlet.http.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/test2")
```

```
6)  public class SessionServlet2 extends HttpServlet
7)  {
8)     public void doGet(HttpServletRequest req,HttpServletResponse res)throws ServletExcep
    tion,IOException
9)     {
10)       PrintWriter out=res.getWriter();
11)       String name = req.getParameter("name");
12)       out.println("<h1>Hi "+name+" Good Morning...");
13)    }
14) }
```

session5
|-login.html
|-WEB-INF
|-classes
|-SessionServlet1.class
|-SessionServlet2.class

Advantages of URL Rewriting:
There is no chance of disabling url rewriting technique. Hence it will work always.

Limitations of URL Rewriting:
1.It is very difficult to rewrite all urls to append session information.Hence it is the most painful technique.

2.URL Rewriting will work only for dynamic documents.

Session Management by using Hidden Form Fields

It is not official technique from SUN Micro Systems.It is just Programmer's trick to remember client information.

In case of Hidden Form Field a hidden (invisible) textfield is used for maintaining the state of an user.

In such case, we store the information in the hidden field, which is required for future purpose.

We can declare hidden form field as follows..

<input type="hidden" name="uname" value="chitu">

Advantage of Hidden Form Field
It will always work whether cookie is disabled or not.

Disadvantage of Hidden Form Field:
1. It is maintained at server side.
2. Extra form submission is required on each page.
3. Session information should be text data

Demo Program for session management by hidden form fields:

login.html:

```
<html>
<body>
<form action="/session7/test1">
<h1>Enter Name<br>
Name:<input type="text" name="uname"><br>
<input type="submit"/></h1>
</form>
</body>
</html>
```

SessionServlet1.java:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;
@WebServlet("/test1")
public class  SessionServlet1 extends HttpServlet
{
   public void doGet(HttpServletRequest req,HttpServletResponse res)throws ServletExcep
tion,IOException
   {
      PrintWriter out=res.getWriter();
      String name=req.getParameter("uname");
      out.println("<h1>Welcome "+name+"<br>");
      out.println("Please provide your age<br>");
      out.println("<form action="/session7/test2">");
      out.println("<input type="hidden" name="uname' value='"+name+"'>");
      out.println("Enter Age:<input type="text" name="age"><br>");
      out.println("<input type="submit" value="submit age">");
      out.println("</form>");
  }
}
```

SessionServlet2.java:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import javax.servlet.annotation.*;
@WebServlet("/test2")
public class SessionServlet2 extends HttpServlet
{
```

```
9)     public void doGet(HttpServletRequest req,HttpServletResponse res)throws ServletExcep
       tion,IOException
10)    {
11)       PrintWriter out=res.getWriter();
12)       String name = req.getParameter("uname");
13)       String age = req.getParameter("age");
14)       out.println("<h1>Hi "+name+" Plz enter your Girl Friend Name...");
15)       out.println("<form action='/session7/test3'>");
16)       out.println("<input type="hidden" name="uname" value='"+name+"'>");
17)       out.println("<input type="hidden" name="uage" value='"+age+"'>");
18)       out.println("Enter Girl Friend Name:<input type="text" name="ugfriend"><br>");
19)       out.println("<input type="submit" value="submit Girl Friend Name">");
20)       out.println("</form>");
21)    }
22) }
```

SessionServlet3.java:

```
1)  import java.io.*;
2)  import javax.servlet.*;
3)  import javax.servlet.http.*;
4)  import java.util.*;
5)  import javax.servlet.annotation.*;
6)  @WebServlet("/test3")
7)  public class SessionServlet3 extends HttpServlet
8)  {
9)     public void doGet(HttpServletRequest req,HttpServletResponse res)
    throws ServletException,IOException
10)    {
11)       PrintWriter out=res.getWriter();
12)       String name = req.getParameter("uname");
13)       String age = req.getParameter("uage");
14)       String gfriend = req.getParameter("ugfriend");
15)       out.println("<h1>Your Total Information is:<br>");
16)       out.println("Name:"+name+"<br/>");
17)       out.println("Age:"+age+"<br/>");
18)       out.println("Girl Friend Name:"+gfriend+"<br/>");
19)       out.println("Thanks for providing Complete information</h1>");
20)    }
21) }
```

```
session7
  |-login.html
  |-WEB-INF
    |-classes
       |-SessionServlet1.class
       |-SessionServlet2.class
       |-SessionServlet3.class
```

Listeners

Objective:

1. Describe web container event life cycle model for the request,session and web application.

2.Create and configure Listener class for each scope
3.Create and configure attribute listener for each scope
4.For the given scenario identify proper attribute listener?

In the web application there may be a chance of occuring several events like

Request object creation
Request object Destruction
Session object creation
Session object Destruction
Context object creation
Context object destruction
Attribute addition in request scope
Attribute Removal in request scope
Attribute Replacement in request scope
.....

whenever these events occur,if we want to do particular operation automatically then we should go for listeners.

i.e Listener listens the events and will perform certain operations automatically.

All Listeners are divided into 3 groups

1. Request Listeners:

These listen events related to request.
There are 2 types of Request Listeners
1. ServletRequestListener
2. ServletRequestAttributeListener

2. Session Listeners:

These listen the events related to session. There are 4 types of session listeners.

1. HttpSessionListener
2. HttpSessionAttributeListener
3. HttpSessionBindingListener
4. HttpSessionActivationListener

3. Context Listeners:

These listen events related to context.

There are 2 types of context listeners
1. ServletContextListener
2. ServletContextAttributeListener.

1. ServletRequestListener(I):
This listener listens life cycle events of request object like creation and destruction.

This interface defines the following 2 methods.

1.public void requestInitialized(ServletRequestEvent e)

This method will be executed automatically at the tine of request object creation. i.e just before starting service() method.

2. public void requestDestroyed(ServletRequestEvent e)

This method will be executed automatically at the time of request object destruction. ie just after completing service() method.

ServletRequestEvent(C):
This class contains the following methods to return request and context objects.

1. public ServletRequest getServletRequest()
2. public ServletContext getServletContext()
ServletRequestEvent is the child class of java.util.EventObject.
EventObject class contains one method getSource()

public Object getSource()
It returns the source which causes the event. In this case web application is the source of event & hence we will get ServletContext object.

Demo Program for ServletRequestListerner:

FirstSevlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  public class FirstSevlet extends HttpServlet
5)  {
6)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
7)     {
8)        PrintWriter out = resp.getWriter();
9)        out.println("<h1>This is RequestListener Demo Servlet</h1><br>");
10)       out.println("<h1>The number of hits for this webapplication is:"+RequestDemoListene
    r.count+"</h1>");
11)    }
12) }
```

RequestDemoListener.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  public class RequestDemoListener implements ServletRequestListener
4)  {
5)     public static int count = 0;
6)
7)     public void requestInitialized(ServletRequestEvent e)
8)     {
9)        count++;
10)       System.out.println("Request Object created at :"+new java.util.Date());
11)       System.out.println("The hit count for this web-application is :"+count);
12)    }
13)    public void requestDestroyed(ServletRequestEvent e)
14)    {
15)       System.out.println("Request Object destroyed :"+new java.util.Date());
16)    }
17) }
```

web.xml:

```
1)  <web-app>
2)
3)  <listener>
4)    <listener-class>RequestDemoListener</listener-class>
5)  </listener>
6)  <servlet>
7)    <servlet-name>FirstSevlet</servlet-name>
8)    <servlet-class>FirstSevlet</servlet-class>
9)  </servlet>
10)
11) <servlet-mapping>
12)   <servlet-name>FirstSevlet</servlet-name>
13)   <url-pattern>/test</url-pattern>
14) </servlet-mapping>
15)
16) </web-app>
```

```
listener1
  |-WEB-INF
    |-web.xml
    |-classes
      |-FirstServlet.class
      |-RequestDemoListener.class
```

Note:

1. We can configure Listener in web.xml by using <listener> tag. <listener> tag is the direct child tag of <web-app> and hence we can take anywhere within <web-app>

2. We can configure more than one listener of same type. The order of execution of these listeners is depends on the order of <listener> tags in web.xml

3. Web container is responsible for the creation of Listener class object.For this web container always calls public no-arg constructor. Hence every listener class should compulsary contains public no-arg constructor.

4. Web container will create Listener object automatically at the time of either server startup or at application deployment.

2. ServletRequestAttributeListener(I):
This listener listens the events related to request scoped attributes like attribute addition,attribute removed and attribute replaced.

This interface defines the following 3 methods

1. public void attributeAdded(ServletRequestAttributeEvent e)
This method will be executed automatically by the web container whenever we are adding an attribute in the request scope.
2. public void attributeRemoved(SRAE e)
3. public void attributeReplaced(SRAE e)

ServletRequestAttributeEvent(C):
This class defines the following 2 methods.

1. public String getName()

Returns the name of the attribute which is added or removed or replaced in the request scope.

2. public Object getValue()
returns the value of the attribute which is added or replaced or removed.

In the case of attribute addition and removal this method returns the corresponding attribute value.But in the case of replacement this method returns old value of the attribute.

Demo Program for SRAL:

FirstSevlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  public class FirstSevlet extends HttpServlet
```

```
5)  {
6)    public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExce
      ption,IOException
7)    {
8)       PrintWriter out = resp.getWriter();
9)       req.setAttribute("durga","scwcd");
10)      req.setAttribute("pavan","scjp");
11)      req.removeAttribute("pavan");
12)      req.setAttribute("durga","scbcd");
13)      out.println("<h1>This is ServletRequestAttributeListener Demo </h1>");
14)
15)   }
16)
17) }
```

RequestAttributeDemoListener.java:

```
1)  import javax.servlet.*;
2)  public class RequestAttributeDemoListener implements ServletRequestAttributeListener
3)  {
4)    public void attributeAdded(ServletRequestAttributeEvent e)
5)    {
6)       System.out.println(e.getName()+"... Attribute Added");
7)    }
8)    public void attributeRemoved(ServletRequestAttributeEvent e)
9)    {
10)      System.out.println(e.getName()+"...Attribute Removed");
11)   }
12)   public void attributeReplaced(ServletRequestAttributeEvent e)
13)   {
14)      System.out.println(e.getName()+"...Attribute Replaced");
15)   }
16)
17) }
```

web.xml:

```
1)  <web-app>
2)
3)  <listener>
4)    <listener-class>RequestAttributeDemoListener</listener-class>
5)  </listener>
6)
7)  <servlet>
8)    <servlet-name>FirstSevlet</servlet-name>
9)    <servlet-class>FirstSevlet</servlet-class>
10) </servlet>
11)
12) <servlet-mapping>
```

```
13)  <servlet-name>FirstSevlet</servlet-name>
14)  <url-pattern>/test</url-pattern>
15) </servlet-mapping>
16) </web-app>
```

listener2
|-WEB-INF
|-web.xml
|-classes
|-FirstServlet.class
|-RequestAttributeDemoListener.class

ServletContextListener:
This listener listens the life cycle events of ServletContext like creation and destruction.

This interface defines the following 2 methods

1. public void contextInitialized(ServletContextEvent e)
This method will be executed automatically by the web container at the time of context object creation. i.e at the time of application deployment or server startup.

2. public void contextDestroyed(ServletContextEvent e)
This method will be executed automatically by the web container at the time of context object destruction. i.e at the time of application undeployment or server shutdown.

ServletContextEvent(C):
It is the child class of java.util.EventObject.
It contains only one method

public ServletContext getServletContext()

Demo program for ServletContextListener to display hitcount of the application where count value will be preserved across Server restarts.

1. RequestDemoListener.java:
Increments count value for every request.

```
1)  import javax.servlet.*;
2)  public class RequestDemoListener implements ServletRequestListener
3)  {
4)     public static int count = 0;
5)     public void requestInitialized(ServletRequestEvent e)
6)     {
7)        count++;
8)     }
9)     public void requestDestroyed(ServletRequestEvent e)
10)    {
11)    }
12) }
```

2. ContextDemoListener.java:

saves the count value to abc.txt file at the time of context object destruction(server shutdown)

It reads the count value from abc.txt file and assign to RequestDemoListener.count variable at context object creation(server startup)

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  public class ContextDemoListener implements ServletContextListener
5)  {
6)     public void contextInitialized(ServletContextEvent e)
7)     {
8)        try{
9)        String path=e.getServletContext().getRealPath("abc.txt");
10)       BufferedReader br = new BufferedReader( new FileReader(path));
11)       String s = br.readLine();
12)       if(s != null)
13)       {
14)          int c = Integer.parseInt(s);
15)          RequestDemoListener.count = c;
16)       }
17)       }
18)       catch(Exception e1){ }
19)    }
20)    public void contextDestroyed(ServletContextEvent e)
21)    {
22)     try{
23)       String path=e.getServletContext().getRealPath("abc.txt");
24)       PrintWriter pw = new PrintWriter(path);
25)       pw.println(RequestDemoListener.count);
26)       pw.flush();
27)      }
28)      catch(Exception e1) {
29)      }
30)    }
31) }
```

3.FirstServlet.java:

To display count value to end user

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  public class FirstSevlet extends HttpServlet
5)  {
```

```
6)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
7)     {
8)        PrintWriter out = resp.getWriter();
9)        out.println("<h1>The number of hits for this webapplication is:"+RequestDemoListene
    r.count+"</h1>");
10)    }
11) }
```

4.web.xml:

For declaring listeners

```
1)  <web-app>
2)
3)  <listener>
4)    <listener-class>RequestDemoListener</listener-class>
5)  </listener>
6)
7)  <listener>
8)    <listener-class>ContextDemoListener</listener-class>
9)  </listener>
10)
11) <servlet>
12)   <servlet-name>FirstSevlet</servlet-name>
13)   <servlet-class>FirstSevlet</servlet-class>
14) </servlet>
15)
16) <servlet-mapping>
17)   <servlet-name>FirstSevlet</servlet-name>
18)   <url-pattern>/test</url-pattern>
19) </servlet-mapping>
20)
21) </web-app>
```

5. abc.txt:

To save count value

```
listenersc
  |-abc.txt
  |-WEB-INF
    |-web.xml
    |-classes
      |-FirstServlet.class
      |-RequestDemoListener.class
      |-ContextDemoListener.class
```

ServletContextAttributeListener(I):

This listener listens the events related to context scoped attributes like addition,removal and replacement.

This interface defines the following 3 methods.
1. public void attributeAdded(ServletContextAttributeEvent e)
2. public void attributeRemoved(SCAE e)
3. public void attributeReplaced(SCAE e)

ServletContextAttributeEvent(C):
It is the child class of ServletContextEvent.
It defines the following 2 methods

1. public String getName()
2. public Object getValue()

HttpSessionListener(I):
This listener listens the life cycle events of HttpSession object like creation and destruction.

This interface defines the following 2 methods

1. public void sessionCreated(HttpSessionEvent e)
2. public void sessionDestroyed(HttpSessionEvent e)

HttpSessionEvent(C):
This class defines only one method getSession()

public HttpSession getSession()

Eg: HS session = req.getSession();

Demo Program to display the number of users currently online:

1. SessionCounter.java:
This listener increments count value for every session object creation and decrements count value for every session object destruction.

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  public class SessionCounter  implements HttpSessionListener
4)  {
5)     static int count = 0;
6)     public void sessionCreated(HttpSessionEvent e)
7)     {
8)        count++;
9)        System.out.println("A new session created with id: "+e.getSession().getId());
10)    }
11)    public void sessionDestroyed(HttpSessionEvent e)
12)    {
13)       count--;
14)       System.out.println("An existing session destroyed with id: "+e.getSession().getId());
15)    }
16) }
```

2. FirstServlet.java

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  public class FirstSevlet extends HttpServlet
5)  {
6)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
7)     {
8)        PrintWriter out = resp.getWriter();
9)        HttpSession s = req.getSession();
10)       s.setMaxInactiveInterval(120);
11)       out.println("<h1>The number of users online is:"+SessionCounter.count+"</h1>");
12)    }
13) }
```

3. web.xml

```
1)  <web-app>
2)
3)  <listener>
4)    <listener-class>SessionCounter</listener-class>
5)  </listener>
6)
7)  <servlet>
8)    <servlet-name>FirstSevlet</servlet-name>
9)    <servlet-class>FirstSevlet</servlet-class>
10) </servlet>
11)
12) <servlet-mapping>
13)   <servlet-name>FirstSevlet</servlet-name>
14)   <url-pattern>/test</url-pattern>
15) </servlet-mapping>
16)
17) </web-app>
```

```
listener3
  |-WEB-INF
    |-web.xml
    |-classes
      |-FirstServlet.class
      |-SessionCounter.class
```

6. HttpSessionAttributeListener(I):

This listener listens the events related to session scoped attributes like attribute addition,removal and replacement.

This interface defines the following 3 methods.

public void attributeAdded(HttpSessionBindingEvent e)
public void attributeRemoved(HSBE e)
public void attributeReplaced(HSBE e)

Note:
There is no event class named with HttpSessionAttributeEvent. For this equivalent event class is HttpSessionBindingEvent.

HttpSessionBindingEvent:

It is the child class of HttpSessionEvent.
This class contains the following 2 methods
public String getName()
public Object getValue()

HttpSessionBindingListener(I):

Whenever a particular type of object adding|removing|replacing in session scope, if we want to perform certain activities then we should go for HttpSessionBindingListener.

This listener defines the following 2 methods

1. public void valueBound(HttpSessionBindingEvent e)
   This method will be executed automatically at the time of attribute addition.

2. public void valueUnbound(HttpSessionBindingEvent e)
   This method will be executed automatically at the time of attribute removal.

Note: In the case of replacement both methods will be executed but valueBound() method first followed by valueUnbound().

Note:
It is not required to configure HttpSessionBindingListener in web.xml. Whenever we are adding an attribute,web container will check the corresponding class implements HttpSessionBindingListener or not. If it implements then the corresponding method will be executed.

If both attribute and binding listeners are configured,then binding listener will be executed first followed by attribute listener.

Demo Program for HttpSessionBindingListener:

Dog.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  public class Dog implements HttpSessionBindingListener
4)  {
5)     public void valueBound(HttpSessionBindingEvent e)
```

```
6)     {
7)        System.out.println("dog object has added to the session scope");
8)     }
9)     public void valueUnbound(HttpSessionBindingEvent e)
10)    {
11)       System.out.println("Dog object has removed from the session scope");
12)    }
13) }
```

SessionAttributeListener.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  public class SessionAttributeListener implements HttpSessionAttributeListener
4)  {
5)     public void attributeAdded(HttpSessionBindingEvent e)
6)     {
7)        System.out.println("attribute added");
8)     }
9)     public void attributeRemoved(HttpSessionBindingEvent e)
10)    {
11)       System.out.println("attribute removed");
12)    }
13)    public void attributeReplaced(HttpSessionBindingEvent e)
14)    {
15)       System.out.println("attribute replaced");
16)    }
17) }
```

web.xml:

```
1)  <web-app>
2)
3)  <listener>
4)   <listener-class>SessionAttributeListener</listener-class>
5)  </listener>
6)
7)   <servlet>
8)    <servlet-name>FirstSevlet</servlet-name>
9)    <servlet-class>FirstSevlet</servlet-class>
10) </servlet>
11)
12) <servlet-mapping>
13)  <servlet-name>FirstSevlet</servlet-name>
14)  <url-pattern>/test</url-pattern>
15) </servlet-mapping>
16)
17) </web-app>
```

Note:
We have to configure only AttributeListener and we are not required to configure binding listener.

FirstServlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  public class FirstSevlet extends HttpServlet
5)  {
6)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
7)     {
8)        PrintWriter out = resp.getWriter();
9)        HttpSession s = req.getSession();
10)       s.setAttribute("a1","SCJP");
11)       s.setAttribute("a1","SCWCD");
12)       s.setAttribute("a2",new Dog());
13)       s.setAttribute("a3",new Dog());
14)       s.setAttribute("a2",new Dog());
15)       s.removeAttribute("a3");
16)       out.println("<h1>This is Session Binding demo</h1>");
17)    }
18)
19) }
```

```
listener4
  |-WEB-INF
    |-web.xml
    |-classes
      |-FirstServlet.class
      |-Dog.class
      |-SessionAttributeListener.class
```

HttpSessionActivationListener:
If functionality is distributed across several JVMs, then that application is called distributed web application.

The main advantages of distributed web applications are
1. By Load balancing we can improve performance of the application
2. By Handling Fail over situations,we can keep our web application robust.(The chance of failure is very very less).

In the distributed applications session object has to migrate from one JVM to another JVM.

Whenever a session object is migrated from one JVM to another JVM,compulsary the corresponding session attributes should be migrated across the network. Hence every session attribute should be compulsary implements Serializable interface.

At the time of session object migration if we want to perform certain activities then we should go for HttpSessionActivationListener.

This interface defines the following 2 methods.

1. public void sessionWillPassivate(HttpSessionEvent e)
This method is called on each implementing object bound to the session just before serialization.

2. public void sessionDidActivate(HttpSessionEvent e)
This method is called on each implementing object bound to the session just after deserialization.

Summary of all Listeners:

Listener	Purpose	Corresponding Methods	Corresponding Events	Corresponding Event Methods	Is required to configure web.xml
Servlet Request Listener	To listens life cycle events of request object. i.e., request object creation & destruction	requestInitialized() requestDestroyed()	ServletRequest Event	getServletRequest() getServletContext()	Yes
Servlet Request Attribute Listener	To listens events related request scoped attributes	attributeAdded() attributeReplaced() attributeRemoved()	ServletRequest AttributeEvent	getName() getValue()	Yes
Servlet Context Listener	To listens life cycle events of context object. i.e., context object creation & destruction	contextInitialized() contextDestroyed()	ServletContext Event	getServletContext()	Yes
Servlet Context Attribute Listener	To listen events related to context scoped attributes	attributeAdded() attributeReplaced() attributeRemoved()	ServletContext AttributeEvent	getName() getValue()	Yes
HttpSession Listener	To listens life cycle events of session object. i.e., session object creation & destruction	sessionCreated() sessionDestroyed()	HttpSession Event	getSession()	Yes

HttpSession Attribute Listener	To listen events related to session scoped attributes	attributeAdded() attributeReplaced() attributeRemoved()	~~HttpSession AttributeEvent~~ HttpSession BindingEvent	getName() getValue()	Yes
HttpSession Binding Listener	When ever we are adding or removing or replacing a particular type of object in session scope , to perform certain activity then we should go for this Listener	valueBound() valueUnbound()	HttpSession BindingEvent	getName() getValue()	not required
HttpSession Activation Listener	If we want to perform any activity just before serialization and just after de-serialization in distributed web-applications	sessionWillPassivate() sessionDidActivate()	HttpSession Event	getSession()	not required

@WebListener Annotation in Servlet 3.0V:

This annotation is replacement for listener configuration in web.xml

Eg:

```
import javax.servlet.annotation.*;
@WebListener
public class RequestDemoListener implements ServletRequestListener
{
 ...
}
```

whenever we are using @WebListener annotation then we are not required to configure listener in web.xml

Unit-5: Web Security

Basic Terminology
Types of Authentication
Declarative Security(web.xml)
Basic Authentication
Form Based Authentication

Programatic Security

Basic Terminology:

Objective: Based on Servlet specification explain the following security mechanisms
1. Authentication
2. Authorization
3. Data Integrity
4. Confidentiality

1. Authentication:
The process of validating the user is called authentication. Usually we can implement authentication by using username and pwd.

Eg: Providing user name and pwd to login into gmail/bank web site etc is called authentication

2. Authorization:
The process of validating access permissions of a user is called authorization. i.e It is the process of checking the whether the user is allowed to access particular resource or not.

After authentication we have to perform authorization.

Usually we can implement authorization by using Access Control List(ACL).

Eg: Eventhough we are valid customer of the bank, we are not authorized to access others account information.We are authorized to access our account information only.

Data Integrity:
It is the process of ensuring that , data should not be changed in trans(transportation) from client to server.

We can implement Data Integrity by using Secure Socket Layer(SSL).

Eg: If we are sending a request to transfer 10000RS from our account to another account, the bank should get request for 10000Rs only but not for 10Lakhs.

Confidentiality:
It is the process of ensuring that no one except intended user is able to understand our information.

We can implement confidentiality by using encryption and decryption algorithms.

Authorization vs Confidentiality:
Authorization prevents the information from reaching unintended users in the first place.

whereas Confidentiality ensures that , even the information falls in the wrong hands, it remains unreadable.

Case Study: Beer web site:

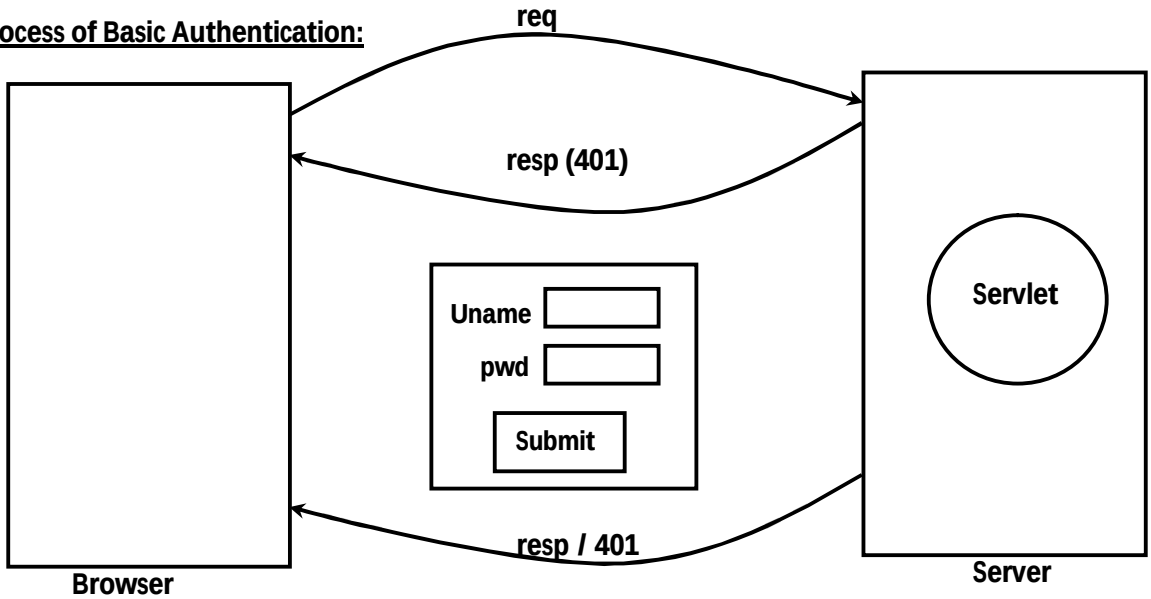
1. Members should have username and pwd to access site
---> Authentication
2. Only Premium Customers are allowed to get 25% Special Discount
--->Authorization
3. Whenever user places an order some sort of confirmation is required-->Data Integrity
4. Whenever a customer makes a purchase, no one allowed to use his credit card information except intended user--->Confidentiality

Types of Authentication:
According to Servlet Specification, there are 4 types of Authentication mechanisms

1. Http Basic Authentication
2. Http Form Based Authentication
3. Http Digest Authentication
4. Https Client Cert Authentication

1. Http Basic Authentication:
It is the simplest and most commonly used Authentication.
This Basic Authentication has introduced in HTTP 1.1 Specification.

Process of Basic Authentication:



Browser sends a request to the server. At this time browser don't know whether requested resource is protected or not. Hence it sends a normal Http Request.

Server will check whether the requested resource is secured or not. If the resource is not secured then server will send the request to that resource and provides proper response to the browser.

If the requested resource is secured, then server will provide 401 status code saying it requires authentication.

By receiving 401 Status code, browser opens a dialogue box prompting for username and pwd.

Once the user enters username and pwd, browser resends the request with user credentials.

Once server receives the request, it validates username and pwd. If valid, then the request will be forwarded to the resource and provide proper response.

If credentials are invalid then server again sends 401 status code.

Advantages of Basic Authentication:
1. It is very easy to implement and set up.
2. All browsers can provide support for this authentication.

Limitations of Basic Authentication:
1. user name and pwd are sending in plain text form from client to server. Hence security is very less in this authentication. This authentication follows Base-64 encoding(no encryption)

2. We cannot customize look and feel of dialogue box.

2. Http Digest Authentication:
This mechanism is exactly same as basic authentication. Except that pwd is sending in encrypted form. Hence this authentication is more secured than Basic Authentication.

Advantage:
When compared with basic authentication, Digest authentication is more secured.

Limitations:
1. Most of the browsers won't provide support for digest authentication, b'z browser is responsible to perform encryption.

2. Most of the servers won't provide support for digest authentication, b'z server is responsible to perform decryption. Even servlet specification does not tell that it is mandatory to provide support for Digest authentication.

3.We cannot change look and feel of the dialogue box.

3. Http Form Based authentication:
It is exactly same as Basic Authentication except that instead of depending on browser's dialogue box, we can provide our own login html form.

Developer is responsible to provide login and error pages, so that we can customize look and feel based on our requirement.

The only requirements of the login form are:
1. The value of action attribute should be j_security_check

2. The form should contains compulsory 2 text fields with the names j_username and j_password

except these all remaining are customizable.

Eg:

```
<form action="j_security_check">
  username: <input type="text" name="j_username">
  pwd:  <input type="text" name="j_password">
  <input type="submit">
</form>
```

Advantages:

1. It is very easy to setup
2. All browsers and all Servers can provide support
3. We can customize look and feel of login form

Limitation:
In this case also, username and pwd are sending in plain text form. Hence security is less.

HTTPS Client Cert Authentication:

Https means HTTP over Secure Socket Layer (SSL).

SSL is a protocol to ensure privacy of sensitive data transferred over the network.

In this mechanism authentication is performed when ssl connection is established b/w client and server.

Total data is transmitted in encryption form by using public key cryptography, which can be handled by both browser and server.

Advantage:
It is the most secured type of authentication

Limitations:
It is very costly to implement and maintain
It require a certificate from 3rd party certificate authority like verisign etc..

Declarative Security(web.xml declarations):
In the deployment descriptor declare
1.<security-constraint>
2.<web-resource>

3.<transport-guarentee>
4.<login-config>
5.<security-role>

We can implement web security by using the following 3 tags.

1. <security-constraint>:
It defines the resources which have to be protected, which roles are allowed to access and security constraint is applicable for which type of http methods etc...

2. <login-config>:
It defines the type of authentication what we are using.

3.<security-role>
It defines the security roles which are allowed in the web application.

Note:
The above 3 tags are direct child tags of <web-app> and hence we can place anywhere within the <web-app>

1. <security-constraint>:
This tag defines the following 3 child tags

1. <web-resource-collection>
Defines the resource which has to be protected

2. <auth-constraint>
Authorization constraint which determines what roles are allowed to access the resource.

3. <user-data-constraint>
It specifies what type of protection is required when trasporting the resource accross the network.

1.<web-resource-collection>:
This tag contains the following 4 child tags

1.<web-resource-name>
2.<description>
3.<url-pattern>
4.<http-method>
It specifies the Http method to which security constraint is applicable.
If we are not using this tag, then security constraint is applicable for all methods.

2.<auth-constraint>:
It specifies which security roles are allowed to access protected resource.

It contains the following 2 child tags.
1.<description>
2.<role-name>

If the security constraint is applicable for all the roles, we have to specify as follows...
<role-name>*</role-name>

3.<user-data-constraint>:
This tag contains the following 2 child tags

1.<description>
2.<transport-guarentee>
This tag specifies what type of guarantee we are providing while transporting the resource across the network.

The allowed values for this tag are:

1.NONE:
It means the data is transported in plain text form.
It is the default value

2.INTEGRAL:
It means the data should not be changed in trans

3. CONFIDENTIAL:
It means the data is transported in encryption form.
The required priority order is: CONFIDENTIAL,INTEGRAL and NONE.

2. <login-config>
This tag specifies the type of authentication we are using.
It contains the following child tags

1.<auth-method>
It specifies the authentication method
The allowed values are
BASIC
DIGEST
FORM
CLIENT-CERT

2. <realm-name>
It specifies the location where we are storing authentication information.
It is required only for basic authentication.

3. <form-login-config>
This tag is required to specify login page url and error page url in the case of Form based authentication.

This tag contains the following 2 child tags
1.<form-login-page> /login.html </form-login-page>
2.<form-error-page> /error.html </form-error-page>

3.<security-role>:

It can be used to define security roles in the web application.
This tag contains the following 2 child tags.

1.<description>
2.<role-name>

Summary of all security related tags:

```
1)  <web-app>
2)          <security-constraint>
3)                  <web-resource-collection>
4)                          <web-resource-name>
5)                          <description>
6)                          <url-pattern>
7)                          <http-method>
8)                  </web-resource-collection>
9)
10)                 <auth-constraint>
11)                         <description>
12)                         <role-name>
13)                 </auth-constraint>
14)
15)                 <user-data-constraint>
16)                   <description>
17)                   <transport-guarentee>
18)                 </user-data-constraint>
19)         </security-constraint>
20)
21)         <login-config>
22)                 <auth-method>
23)                 <realm-name>
24)                 <form-login-config>
25)                   <form-login-page>
26)                   <form-error-page>
27)                 </form-login-config>
28)         </login-config>
29)
30)          <security-role>
31)            <description>
32)                   <role-name>
33)          </security-role>
34)
35) </web-app>
```

Demo Program for Basic Authentication:

web.xml:

```
1)  <web-app>
2)  <servlet>
3)    <servlet-name>FirstSevlet</servlet-name>
4)    <servlet-class>FirstServlet</servlet-class>
5)  </servlet>
6)
7)  <servlet-mapping>
8)    <servlet-name>FirstSevlet</servlet-name>
9)    <url-pattern>/test</url-pattern>
10) </servlet-mapping>
11)
12) <security-constraint>
13)   <web-resource-collection>
14)       <web-resource-name>CheckedServlet</web-resource-name>
15)     <url-pattern>/test</url-pattern>
16)    <http-method>POST</http-method>
17)    <http-method>GET</http-method>
18)  </web-resource-collection>
19)
20)   <auth-constraint>
21)     <role-name>durgarole</role-name>
22)   </auth-constraint>
23)
24) </security-constraint>
25)
26) <login-config>
27)   <auth-method>BASIC</auth-method>
28) </login-config>
29)
30)
31)  <security-role>
32)    <role-name>durgarole</role-name>
33)  </security-role>
34)
35) </web-app>
```

tomcat-users.xml:

```
<tomcat-users>
 ...
<role rolename="durgarole"/>
<user name="durga" password="java" roles="durgarole" />
<user name="ravi" password="scjp" roles="durgarole" />
</tomcat-users>
```

postreqform.html:

```
1)  <html>
2)  <body><h1> Basic Authentication Demo to send POST request</h1>
3)  <form action = "/webs1/test" method="POST">
4)  Enter Text :<input type=text name="text">
5)  <input type=submit>
6)  </form>
7)  </body>
8)  </html>
```

FirstServlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  public class FirstServlet extends HttpServlet
5)  {
6)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
7)     {
8)        PrintWriter out = resp.getWriter();
9)        out.println("<h1>Get:After Authentication only we can access this servlet</h1>");
10)    }
11)    public void doPost(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
12)    {
13)       PrintWriter out = resp.getWriter();
14)       out.println("<h1>POST:After Authentication only we can access this servlet</h1>");
15)    }
16) }
```

Demo Program for FORM-BASED Authentication:

login.html:

```
1)  <html>
2)  <body><h1> Welcome to Durga Software Solutions</h1><br>
3)  <h2>Please login to avail the facilities....</h2>
4)  <form action = "j_security_check">
5)  Enter Name :<input type=text name="j_username"><br>
6)  Enter password :<input type=password name="j_password"><br>
7)  <input type=submit>
8)  </form>
9)  </body>
10) </html>
```

error.html:

```
<h1>Your credentials are not correct. Please provide valid credentials</h1>
```

web.xml:

```
1)  <web-app>
2)  <servlet>
3)    <servlet-name>FirstSevlet</servlet-name>
4)    <servlet-class>FirstSevlet</servlet-class>
5)  </servlet>
6)
7)  <servlet-mapping>
8)    <servlet-name>FirstSevlet</servlet-name>
9)    <url-pattern>/test</url-pattern>
10) </servlet-mapping>
11)
12) <security-constraint>
13)   <web-resource-collection>
14)     <web-resource-name>CheckedServlet</web-resource-name>
15)     <url-pattern>/test</url-pattern>
16)     <http-method>GET</http-method>
17)     <http-method>POST</http-method>
18) </web-resource-collection>
19) <auth-constraint>
20)   <role-name>durgarole</role-name>
21) </auth-constraint>
22) </security-constraint>
23)
24) <login-config>
25)  <auth-method>FORM</auth-method>
26)   <form-login-config>
27)       <form-login-page>/login.html</form-login-page>
28)       <form-error-page>/error.html</form-error-page>
29)   </form-login-config>
30) </login-config>
31)
32)  <security-role>
33)  <role-name>durgarole</role-name>
34)  </security-role>
35)  </web-app>
```

tomcat-users.xml:

```
1)  <tomcat-users>
2)   ...
3)     <role rolename="durgarole"/>
4)     <user name="durga" password="java" roles="durgarole" />
5)     <user name="ravi" password="scjp" roles="durgarole" />
6)  </tomcat-users>
```

postreqform.html:

```
1)  <html>
2)  <body><h1> Basic Authentication Demo to send POST request</h1>
3)  <form action = "/webs2/test" method="POST">
4)  Enter Text :<input type=text name="text">
5)  <input type=submit>
6)  </form>
7)  </body>
8)  </html>
```

FirstServlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  public class FirstServlet extends HttpServlet
5)  {
6)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
7)     {
8)        PrintWriter out = resp.getWriter();
9)        out.println("<h1>Get:After Authentication only we can access this servlet</h1>");
10)    }
11)    public void doPost(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
12)    {
13)       PrintWriter out = resp.getWriter();
14)       out.println("<h1>POST:After Authentication only we can access this servlet</h1>");
15)    }
16) }
```

Programmatic Security:

Sometimes declarative security is not enough, compulsory we should go for programmatic security.

Based on the user role, we have to provide the corresponding response. If the user is admin then admin related response and if the user is manager then manager related response we have to provide.

For this type of requirement compulsory we should go for Programmatic Security.

We can implement programmatic security by using the following methods of HttpServletRequest.

1. public boolean isUserInRole(String rolename)

If the authenticated user belongs to the specified role then this method returns "true"

If the authenticated user not belongs to the specified role or if the user not authenticated then this method returns false.

2. public String getRemoteUser()

Returns the authenticated user name(login name)

If the user has not been authenticated then this method returns null.

3. public Principal getUserPrincipal()

Returns java.security.Principal object which contains user name.

Returns null if the user has not been authenticated.

Eg:

```
1) if(req.isUserInRole("admin"))
2) {
3)   out.println("Admin related response");
4) }
5) else
6) {
7)   out.println("non admin related response");
8) }
```

The main problem in this approach is we are hard coding the role names in the servlet.

If there is any change in the role-name, modifying servlet code is costly and creates maintanence problems.

To overcome this problem we have to use <security-role-ref> tag. By using this tag we can map hard coded role names with original role name.

```
1) <servlet>
2)    .....
3)    <security-role-ref>
4)       <role-name>admin</role-name>
5)       <role-link>durgaadmin</role-link>
6)    </security-role-ref>
7) </servlet>
```

where admin is logical role name and durgaadmin is original role name.

Demo Program for programmatic Security:

web.xml:

```
1) <web-app>
2) <servlet>
3)  <servlet-name>FirstSevlet</servlet-name>
4)  <servlet-class>FirstSevlet</servlet-class>
5)  <security-role-ref>
6)     <role-name>hero</role-name>
7)     <role-link>durgaadmin</role-link>
8)  </security-role-ref>
```

```
9)  </servlet>
10)
11) <servlet-mapping>
12)  <servlet-name>FirstSevlet</servlet-name>
13)  <url-pattern>/test</url-pattern>
14) </servlet-mapping>
15)
16) <security-constraint>
17)  <web-resource-collection>
18)   <web-resource-name>CheckedServlet</web-resource-name>
19)   <url-pattern>/test</url-pattern>
20)   <http-method>GET</http-method>
21) </web-resource-collection>
22) <auth-constraint>
23)  <role-name>durgaadmin</role-name>
24)  <role-name>durgamanager</role-name>
25) </auth-constraint>
26) </security-constraint>
27)
28) <login-config>
29)  <auth-method>BASIC</auth-method>
30) </login-config>
31)
32) <security-role>
33)  <role-name>durgaadmin</role-name>
34) </security-role>
35)
36)  <security-role>
37)  <role-name>durgamanager</role-name>
38)  </security-role>
39)  </web-app>
```

tomcat-users.xml:

```
1)  <tomcat-users>
2)   ...
3)     <role rolename="durgaadmin"/>
4)     <role rolename="durgamanager"/>
5)     <user name="pawan" password="kalyan" roles="durgaadmin" />
6)     <user name="shiva" password="scwcd" roles="durgaadmin" />
7)     <user name="mahesh" password="babu" roles="durgamanager" />
8)  </tomcat-users>
```

login.html:

```
1)  <html>
2)  <body><h1> Programatic Security To send Post Request</h1>
3)  <form action = "/webs3/test" method="POST">
4)  Enter Text :<input type=text name=uname>
```

```
5)  <input type=submit>
6)  </form>
7)  </body>
8)  </html>
```

FirstServlet.java:

```
1)  import javax.servlet.*;
2)  import javax.servlet.http.*;
3)  import java.io.*;
4)  public class FirstSevlet extends HttpServlet
5)  {
6)     public void doGet(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
7)     {
8)        PrintWriter out = resp.getWriter();
9)        String name = req.getRemoteUser();
10)       out.println("<h1>Hi .."+name+"</h1><br>");
11)       if(req.isUserInRole("hero"))
12)       {
13)          out.println("<h1>This is Hero Home Page</h1>");
14)       }
15)       else
16)       {
17)          out.println("<h1>This is Others Home Page</h1>");
18)       }
19)    }
20)    public void doPost(HttpServletRequest req,HttpServletResponse resp) throws ServletExc
    eption,IOException
21)    {
22)       doGet(req,resp);
23)    }
24) }
```

OCWCD

Question Bank

Unit-1: The Servlet Technology Model

Objectives

For each of the HTTP Methods (such as GET, POST, HEAD, and so on) describe the purpose of the method and the technical characteristics of the HTTP Method protocol, list triggers that might cause a Client (usually a Web browser) to use the method; and identify the HttpServlet method that corresponds to the HTTP Method.

Using the HttpServletRequest interface, write code to retrieve HTML form parameters from the request, retrieve HTTP request header information, or retrieve cookies from the request.

Using the HttpServletResponse interface, write code to set an HTTP response header, set the content type of the response, acquire a text stream for the response, acquire a binary stream for the response, redirect an HTTP request to another URL, or add cookies to the response.

Describe the purpose and event sequence of the servlet life cycle: (1) servlet class loading, (2) servlet instantiation, (3) call the init method, (4) call the service method, and (5) call destroy method.

Q1. You are creating a web form with this HTML:
11. <form action="sendOrder.jsp">
12. <input type="text" name="creditCard">
13. <input type="text" name="expirationDate">
14. <input type="submit">
15. </form>

Which HTTP method is used when sending this request from the browser?
A. GET **B. PUT**
C. POST **D. SEND**
E. FORM
Answer: A

Q2. Given a header in an HTTP request: X-Retries: 4

Which two retrieve the value of the header from a given HttpServletRequest request? (Choose two.)
A. request.getHeader("X-Retries")
B. request.getIntHeader("X-Retries")
C. request.getRequestHeader("X-Retries")
D. request.getHeaders("X-Retries").get(0)
E. request.getRequestHeaders("X-Retries").get(0)
Answer: A, B

Q3. For a given ServletResponse response, which two retrieve an object for writing text data? (Choose two.)

A. response.getWriter()

B. response.getOutputStream()
C. response.getOutputWriter()
D. response.getWriter().getOutputStream()
E. response.getWriter(Writer.OUTPUT_TEXT)

Answer: A, B

Q4. Given an HttpServletRequest request and HttpServletResponse response,

which sets a cookie "username" with the value "joe" in a servlet?
A. request.addCookie("username", "joe")
B. request.setCookie("username", "joe")
C. response.addCookie("username", "joe")
D. request.addHeader(new Cookie("username", "joe"))
E. request.addCookie(new Cookie("username", "joe"))
F. response.addCookie(new Cookie("username", "joe"))
G. response.addHeader(new Cookie("username", "joe"))
Answer: F

Q5. Your web page includes a Java SE v1.5 applet with the following declaration:
11. <object classid='clsid:CAFEEFAC-0015-0000-0000-ABCDEFFEDCBA'
12. width='200' height='200'>
13. <param name='code' value='Applet.class' />
14. </object>

Which HTTP method is used to retrieve the applet code?
A. GET B. PUT
C. POST D. RETRIEVE

Answer: A

Q6. You are creating a servlet that generates stock market graphs. You want to provide the web browser with precise information about the amount of data being sent in the response stream.

Which two HttpServletResponse methods will you use to provide this information? (Choose two.)
A. response.setLength(numberOfBytes);
B. response.setContentLength(numberOfBytes);
C. response.setHeader("Length", numberOfBytes);
D. response.setIntHeader("Length", numberOfBytes);
E. response.setHeader("Content-Length", numberOfBytes);
F. response.setIntHeader("Content-Length", numberOfBytes);

Answer: B, F

Q7. You need to retrieve the username cookie from an HTTP request. If this cookie does NOT exist, then the c variable will be null.

Which code snippet must be used to retrieve this cookie object?

A. 10. Cookie c = request.getCookie("username");
B. 10. Cookie c = null;

11. for (Iterator i = request.getCookies();
12. i.hasNext();) {
13. Cookie o = (Cookie) i.next();
14. if (o.getName().equals("username")) {
15. c = o;
16. break;
17. }
18. }
C. 10. Cookie c = null;
11. for (Enumeration e = request.getCookies();
12. e.hasMoreElements();) {
13. Cookie o = (Cookie) e.nextElement();
14. if (o.getName().equals("username")) {
15. c = o;
16. break;
17. }
18. }
D. 10. Cookie c = null;
11. Cookie[] cookies = request.getCookies();
12. for (int i = 0; i < cookies.length; i++) {
13. if (cookies[i].getName().equals("username")) {
14. c = cookies[i];
15. break;
16. }
17. }

Answer: D

Q8. Given:
10. public void service(ServletRequest request,
11. ServletResponse response) {
12. ServletInputStream sis =
13. // insert code here
14. }

Which retrieves the binary input stream on line 13?

A. request.getWriter();
B. request.getReader();
C. request.getInputStream();
D. request.getResourceAsStream();
E. request.getResourceAsStream(ServletRequest.REQUEST);
Answer: C

Q9. Click the Exhibit button.

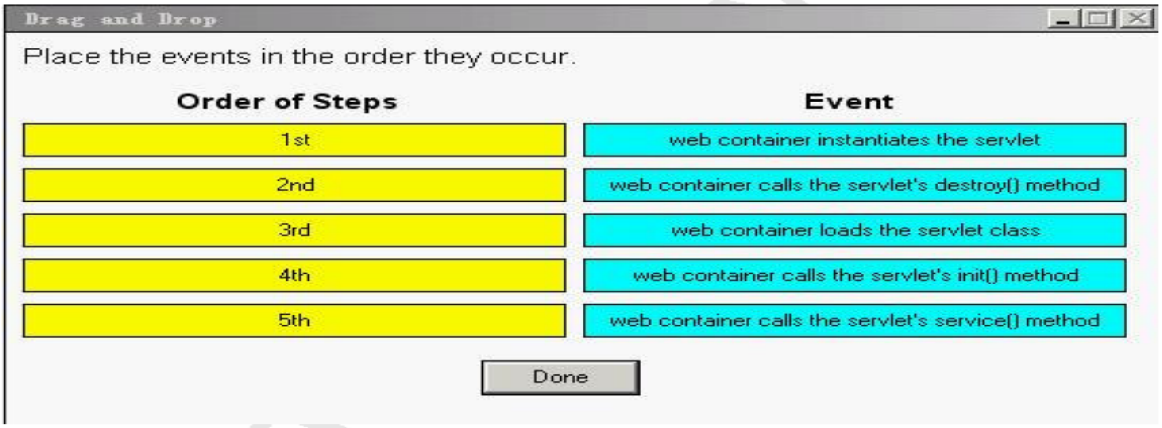
As a maintenance feature, you have created this servlet to allow you to upload and remove files on your web server. Unfortunately, while testing this servlet, you try to upload a file using an HTTP request and on this servlet, the web container returns a 404 status.
What is wrong with this servlet?

```
 1. package com.example;
 2.
 3. import javax.servlet.http.*;
 4.
 5. public class MyWebDAV extends HttpServlet
{
 6.   private String resourceDirectory;
 7.
 8.   public MyWebDAV(String resDir) {
 9.     this.resourceDirectory = resDir;
10.   }
11.   public void doPut(HttpServletRequest
req,
12.                     HttpServletResponse
resp) {
13.     // store file to resourceDirectory
(code not shown)
20.   }
21.   public void doDelete(HttpServletRequest
req,
22.
HttpServletResponse resp) {
23.     // remove file from resourceDirectory
(code not shown)
30.   }
31. }
```

A. HTTP does NOT support file upload operations.
B. The servlet constructor must NOT have any parameters.
C. The servlet needs a service method to dispatch the requests to the helper methods.
D. The doPut and doDelete methods do NOT map to the proper HTTP methods.

Answer: B

Q10. Click the Task button.
Place the events in the order they occur.



Answer:
1st:web container loads the servlet class.
2nd:web container instantiates the servlet
3rd. web container calls the servlets init() method.
4th: web container calls the servlets service() method.'
5th: web container calls the servlets destroy() methos.

Q11. For an HttpServletResponse response, which two create a custom header?(Choose two.)

A. response.setHeader("X-MyHeader", "34");
B. response.addHeader("X-MyHeader", "34");
C. response.setHeader(new HttpHeader("X-MyHeader", "34"));
D. response.addHeader(new HttpHeader("X-MyHeader", "34"));
E. response.addHeader(new ServletHeader("X-MyHeader", "34"));
F. response.setHeader(new ServletHeader("X-MyHeader", "34"));

Answer: A, B

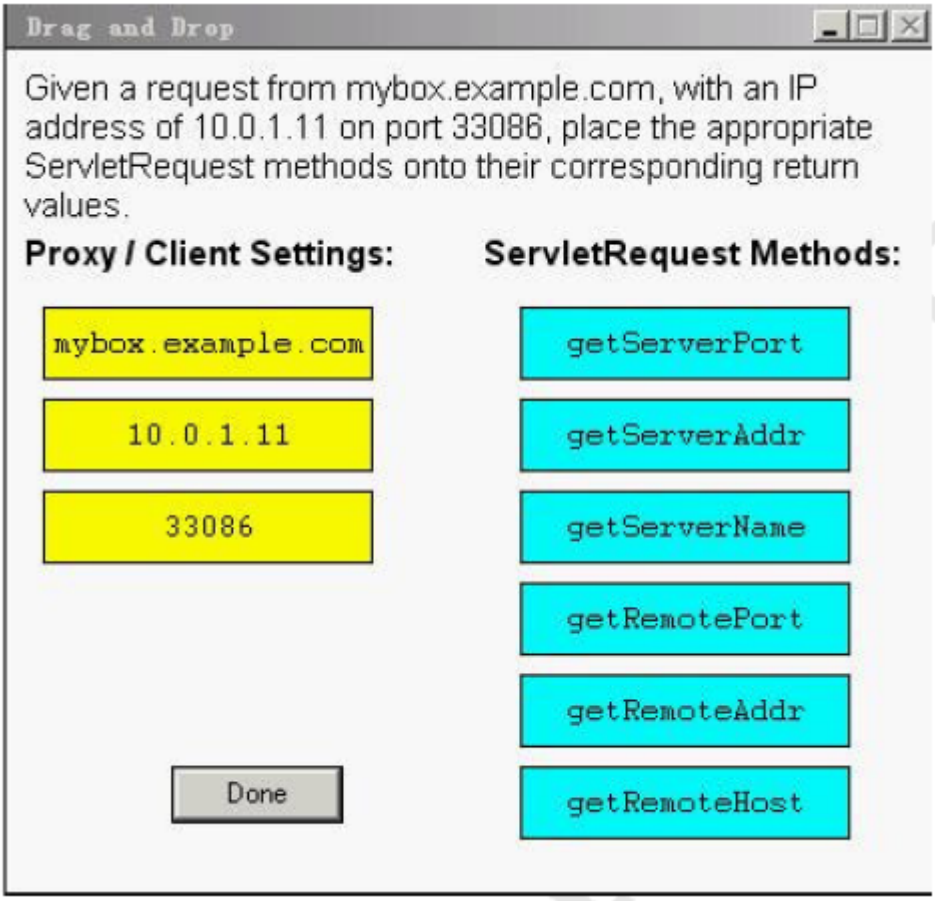
Q12 .You need to create a servlet filter that stores all request headers to a database for all requests to the web application's home page "/index.jsp". Which HttpServletRequest method allows you to retrieve all of the request headers?

A. String[] getHeaderNames()
B. String[] getRequestHeaders()
C. java.util.Iterator getHeaderNames()
D. java.util.Iterator getRequestHeaders()
E. java.util.Enumeration getHeaderNames()
F. java.util.Enumeration getRequestHeaders()

Answer: E

Q13. Click the Task button.

Given a request from mybox.example.com, with an IP address of 10.0.1.11 on port 33086, place theappropriate ServletRequest methods onto their corresponding return values.



Answers:

Mybox.example.com : getRemoteHost
10.0.1.11 : getRemoteAddr

Q: 14 Your web application requires the ability to load and remove web files dynamically to the web container's file system. Which two HTTP methods are used to perform these actions? (Choose two.)

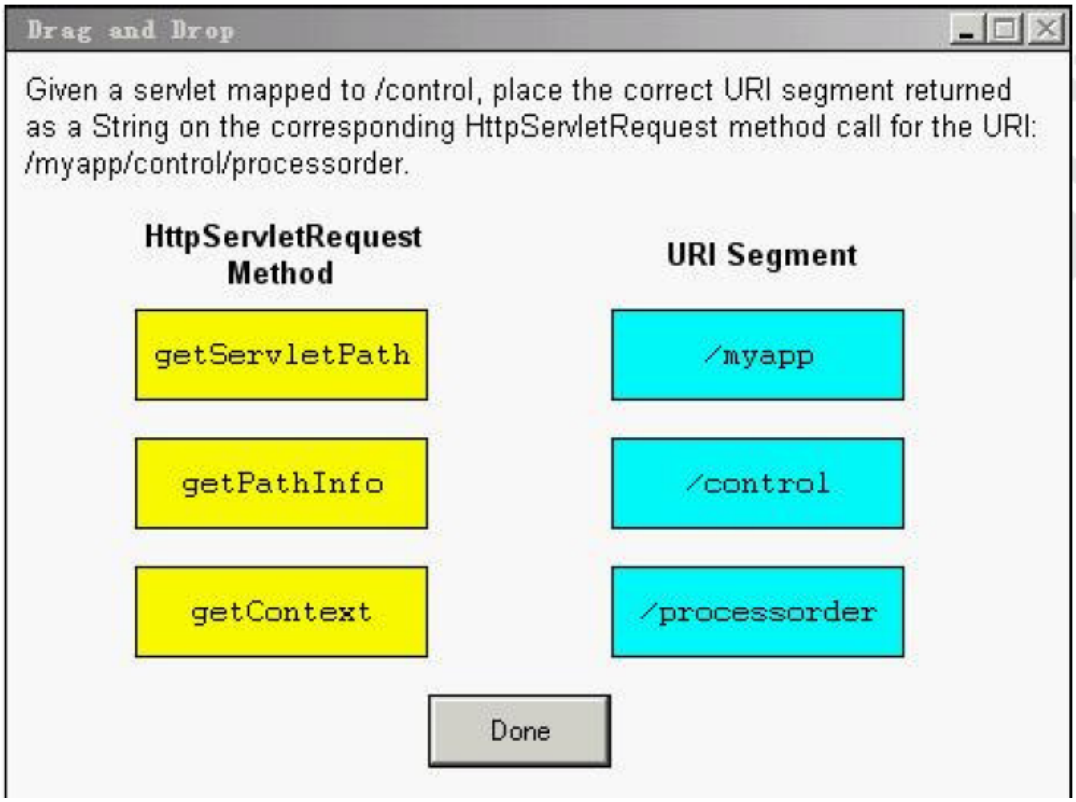
A. PUT
B. POST
C. SEND
D. DELETE
E. REMOVE
F. DESTROY

Answer: A, D

Q15. Which retrieves all cookies sent in a given HttpServletRequest request?

A. request.getCookies()
B. request.getAttributes()
C. request.getSession().getCookies()
D. request.getSession().getAttributes()
Answer: A

Q16. Click the Task button.
Given a servlet mapped to /control, place the correct URI segment returned as a String on the corresponding HttpServletRequest method call for the URI: /myapp/control/processorder.



Answers:
getServletpath( ) --------/processorder
getPathInfo ( ) ----- /Control
getContext ( ) ----- /myapp

Q17. A web browser need NOT always perform a complete request for a particular page that it suspects might NOT have changed. The HTTP specification provides a mechanism for the browser to retrieve only a partial response from the web server; this response includes information, such as the Last-Modified date but NOT the body of the page. Which HTTP method will the browser use to retrieve such a partial response?

A. GET
B. ASK
C. SEND
D. HEAD
E. TRACE
F. OPTIONS
Answer: D

Q 18. You are creating a servlet that generates stock market graphs. You want to provide the web browser with precise information about the amount of data being sent in the response stream. Which two HttpServletResponse methods will you use to provide this information? (Choose two.)

A. response.setLength(numberOfBytes);
B. response.setContentLength(numberOfBytes);
C. response.setHeader("Length", numberOfBytes);
D. response.setIntHeader("Length", numberOfBytes);
E. response.setHeader("Content-Length", numberOfBytes);
F. response.setIntHeader("Content-Length", numberOfBytes);

Answer: B, F

Q19. Which two prevent a servlet from handling requests? (Choose two.)

A. The servlet's init method returns a non-zero status.
B. The servlet's init method throws a ServletException.
C. The servlet's init method sets the ServletResponse's content length to 0.
D. The servlet's init method sets the ServletResponse's content type to null.
E. The servlet's init method does NOT return within a time period defined by the servlet container.

Answer: B, E

Unit- 2: The Structure and Deployment of Web Applications

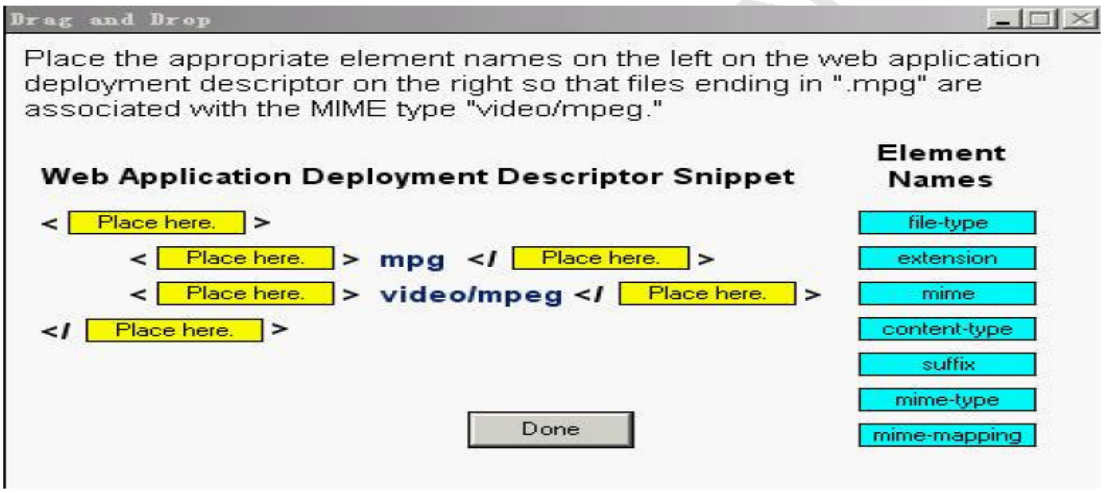
Objectives

1. Construct the file and directory structure of a Web Application that may contain (a) static content, (b) JSP pages, (c) servlet classes, (d) the deployment descriptor, (e) tag libraries, (d) JAR files, and (e) Java class files; and describe how to protect resource files from HTTP access.
2. Describe the purpose and semantics of the deployment descriptor.
3. Construct the correct structure of the deployment descriptor.
Explain the purpose of a WAR file and describe the contents of a WAR file, how one may be constructed

Q1 Which three are valid URL mappings to a servlet in a web deployment descriptor? (Choose three.)

A. */* **B. *.do**
C. MyServlet **D. /MyServlet**
E. /MyServlet/* **F. MyServlet/*.jsp**
Answer: B, D, E

Q2. Click the Task button.
Place the appropriate element names on the left on the web application deployment descriptor on the right so that files ending in ".mpg" are associated with the MIME type "video/mpeg."



Answers:
<mime-mapping>
<mime-type>mpg</mime-type>
<extension>vedio/mpeg</extension>
</mime-mapping>

Q3. Which three web application deployment descriptor elements allow webcomponents to gain references to resources or EJB components? (Choose three.)

A. ejb-ref **B. jdbc-ref**

C. servlet-ref
D. resource-ref
E. javamail-ref
F. ejb-remote-ref
G. resource-env-ref
Answer: A, D, G

Q4. Which two actions protect a resource file from direct HTTP access within a web application? (Choose two.)

A. placing it in the /secure directory
B. placing it in the /WEB-INF directory
C. placing it in the /META-INF/secure directory
D. creating a <web-resource> element within the deployment descriptor
E. creating a <secure-resource> element within the deployment descriptor
Answer: B, C

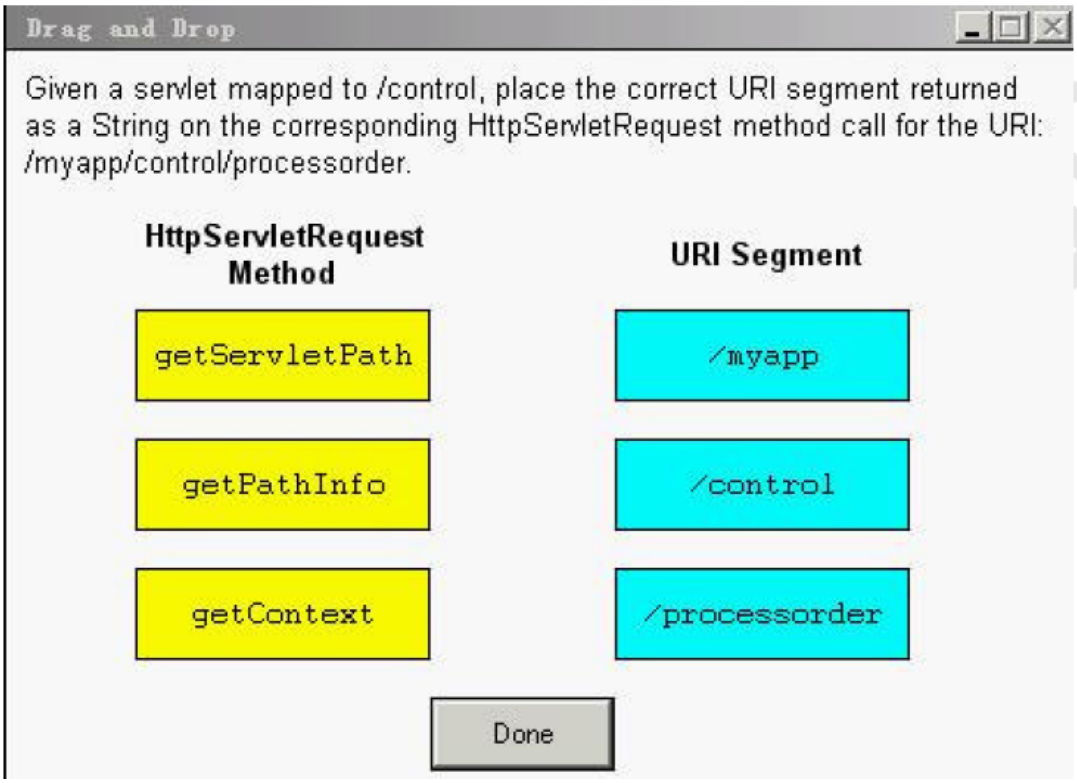
Q5. Given that www.example.com/SCWCDtestApp is a validly deployed Java EE web application and that all of the JSP files specified in the requests below exist in the locations specified.Which two requests, issued from a browser, will return an HTTP 404 error? (Choose two.)

A. http://www.example.com/SCWCDtestApp/test.jsp
B. http://www.example.com/SCWCDtestApp/WEB-INF/test.jsp
C. http://www.example.com/SCWCDtestApp/WEB-WAR/test.jsp
D. http://www.example.com/SCWCDtestApp/Customer/test.jsp
E. http://www.example.com/SCWCDtestApp/META-INF/test.jsp
F. http://www.example.com/SCWCDtestApp/Customer/Update/test.jsp
Answer: B, E

Q6. Which two about WAR files are true? (Choose two.)
A. WAR files must be located in the web application library directory.
B. WAR files must contain the web application deployment descriptor.
C. WAR files must be created by using archive tools designed specifically for that purpose.
D. The web container must serve the content of any META-INF directory located in a WAR file.
E. The web container must allow access to resources in JARs in the web application library directory.

Answer: B, E

Q7.Click the Task button.
Given a servlet mapped to /control, place the correct URI segment returned as a String on thecorresponding HttpServletRequest method call for the URI: /myapp/control/processorder.



Answers:
getServletPath()--->/control
getPathInfo() --->/processorder
getContext()---->/myapp

Q8. Given an HttpSession session, a ServletRequest request, and a ServletContext context, which retrieves a URL to /WEB-INF/myconfig.xml within a web application?
A. session.getResource("/WEB-INF/myconfig.xml")
B. request.getResource("/WEB-INF/myconfig.xml")
C. context.getResource("/WEB-INF/myconfig.xml")
D. getClass().getResource("/WEB-INF/myconfig.xml")

Answer: C
Q9. You have built a web application with tight security. Several directories ofyour webapp are used for internal purposes and you have overridden the default servlet to send an HTTP403 status code for any request that maps to one of these directories. During testing, the QualityAssurance director decided that they did NOT like seeing the bare response page generated by Firefoxand Internet Explorer. The director recommended that the webapp should return a more user-friendlyweb page that has the same look-and-feel as the webapp plus links to the webapp's search engine. Youhave created this JSP page in the /WEB- INF/jsps/error403.jsp file. You do NOT want to alter the complex logic of the default servlet. How can you declare that the web container must send this JSP page whenever a 403 status is generated?
A. <error-page>
<error-code>403</error-code>
<url>/WEB-INF/jsps/error403.jsp</url>
</error-page>
B. <error-page>

<status-code>403</status-code>
<url>/WEB-INF/jsps/error403.jsp</url>
</error-page>
C. <error-page>
<error-code>403</error-code>
<location>/WEB-INF/jsps/error403.jsp</location>
</error-page>
D. <error-page>
<status-code>403</status-code>
<location>/WEB-INF/jsps/error403.jsp</location>
</error-page>
Answer: C

Q: 10 You want to create a valid directory structure for your Java EE web application, and your application uses tag files and a JAR file. Which three must be located directly in your WEB-INF directory (NOT in a subdirectory of WEB-INF)? (Choose three.)

A. The JAR file
B. A directory called lib
C. A directory called tags
D. A directory called TLDs
E. A directory called classes
F. A directory called META-INF
Answer: B, C, E

Q 11. Given:
11. public class MyServlet extends HttpServlet {
12. public void service(HttpServletRequest request,
13. HttpServletResponse response)
14. throws ServletException, IOException {
15. // insert code here
16. }
17. }
and this element in the web application's deployment descriptor:

<error-page>
<error-code>302</error-code>
<location>/html/error.html</location>
</error-page>

Which, inserted at line 15, causes the container to redirect control to the error.html resource?

A. response.setError(302);
B. response.sendError(302);
C. response.setStatus(302);
D. response.sendRedirect(302);
E. response.sendErrorRedirect(302);
Answer: B

Q12. Which element of the web application deployment descriptor defines the servlet class associated with a servlet instance?
A. <class>
B. <webapp>
C. <servlet>
D. <codebase>
E. <servlet-class>
F. <servlet-mapping>
Answer: E

Q13. Within the web application deployment descriptor, which defines a valid JNDI environment entry?
A. <env-entry>
<env-entry-type>java.lang.Boolean</env-entry-type>
<env-entry-value>true</env-entry-value>
</env-entry>
B. <env-entry>
<env-entry-name>param/MyExampleString</env-entry-name>
<env-entry-value>This is an Example</env-entry-value>
</env-entry>
C. <env-entry>
<env-entry-name>param/MyExampleString</env-entry-name>
<env-entry-type>int</env-entry-type>
<env-entry-value>10</env-entry-value>
</env-entry>
D. <env-entry>
<env-entry-name>param/MyExampleString</env-entry-name>
<env-entry-type>java.lang.String</env-entry-type>
<env-entry-value>This is an Example</env-entry-value>
</env-entry>
Answer: D

Q14. Which three are described in the standard web application deployment descriptor? (Choose three.)
A. session configuration
B. MIME type mappings
C. context root for the application
D. servlet instance pool configuration
E. web container default port bindings
F. ServletContext initialization parameters
Answer: A, B, F

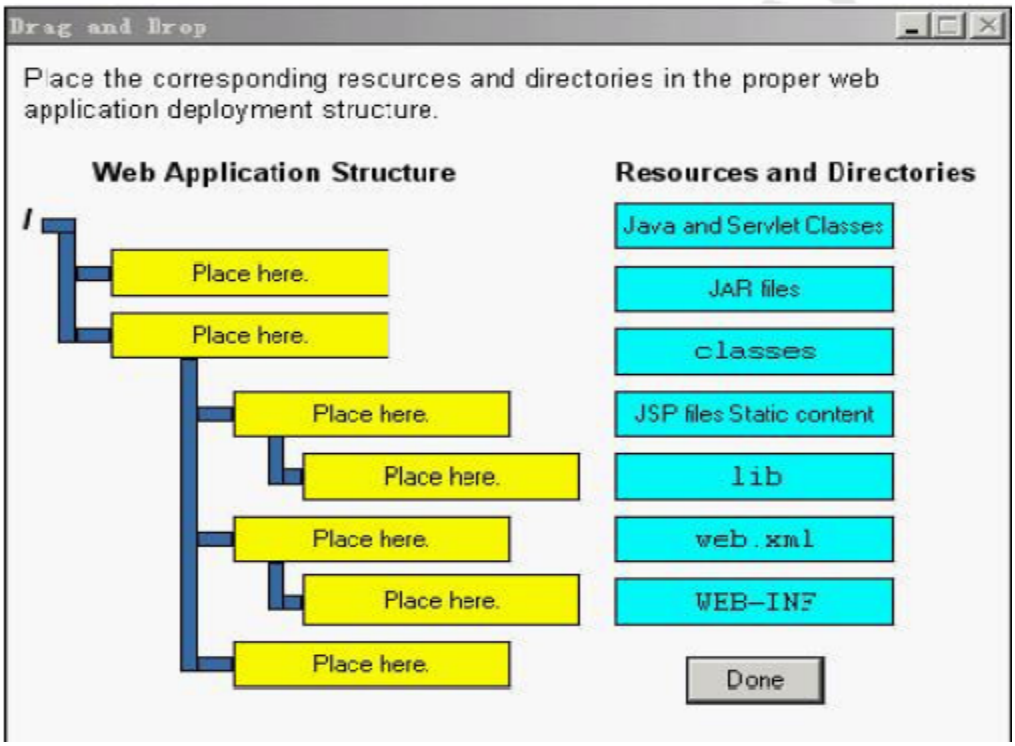
Q15. Which two are true regarding a web application class loader?
(Choose two.)
A. A web application may override the web container's implementation classes.
B. A web application running in a J2EE product may override classes in the javax.* namespace.
C. A web application class loader may NOT override any classes in the java.* and javax.* namespaces.

D. Resources in the WAR class directory or in any of the JAR files within the library directory may be accessed using the J2SE semantics of getResource.
E. Resources in the WAR class directory or in any of the JAR files within the library directory CANNOT be accessed using the J2SE semantics of getResource.

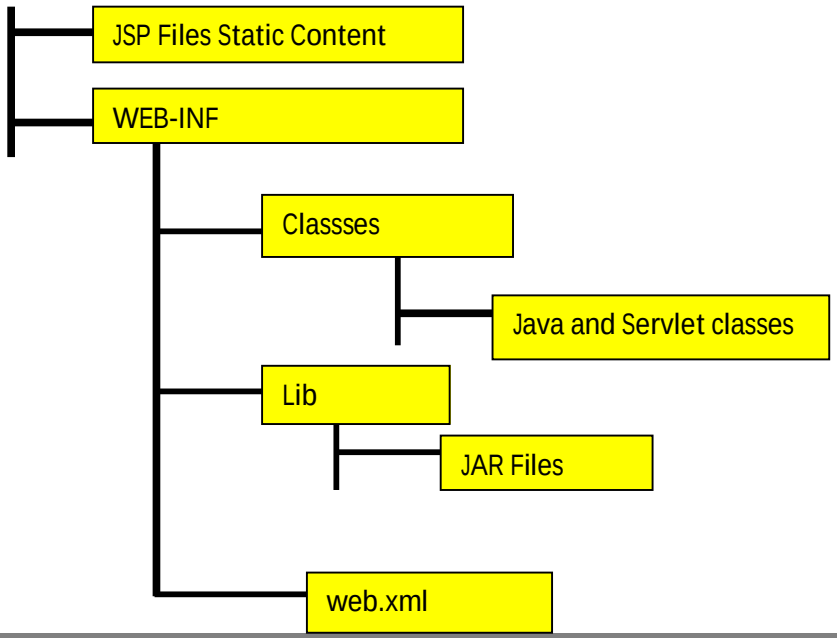
Answer: C, D

Q16. Click the Task button.

Place the corresponding resources and directories in the proper web application deployment structure.



Answers:



Q17. You want to create a valid directory structure for your Java EE web application, and you want to put your web application into a WAR file called MyApp.war. Which two are true about the WAR file? (Choose two.)

A. At deploy time, Java EE containers add a directory called META-INF directly into the MyApp directory.
B. At deploy time, Java EE containers add a file called MANIFEST.MF directly into the MyApp directory.
C. It can instruct your Java EE container to verify, at deploy time, whether you have properly configured your application's classes.
D. At deploy time, Java EE containers add a directory call META-WAR directly into the MyApp directory.

Answer: A, C

Q18. Which two from the web application deployment descriptor are valid?(Choose two.)

A. <error-page>
<exception-type>*</exception-type>
<location>/error.html</location>
</error-page>
B. <error-page>
<exception-type>java.lang.Error</exception-type>
<location>/error.html</location>
</error-page>
C. <error-page>
<exception-type>java.lang.Throwable</exception-type>
<location>/error.html</location>
</error-page>
D. <error-page>
<exception-type>java.io.IOException</exception-type>
<location>/error.html</location>
</error-page>
E. <error-page>
<exception-type>NullPointerException</exception-type>
<location>/error.html</location>
</error-page>

Answer: C, D

Q19. After a merger with another small business, your company has inherited a legacy WAR file but the original source files were lost. After reading the documentation of that web application, you discover that the WAR file contains a useful tag library that you want to reuse in your own webapp packaged as a WAR file.What do you need to do to reuse this tag library?

A. Simply rename the legacy WAR file as a JAR file and place it in your webapp's library directory.
B. Unpack the legacy WAR file, move the TLD file to the META-INF directory, repackage the whole thing as a JAR file, and place that JAR file in your webapp's library directory.

C. Unpack the legacy WAR file, move the TLD file to the META-INF directory, move the class files to the top-level directory, repackage the whole thing as a JAR file, and place that JAR file in your webapp's library directory.
D. Unpack the legacy WAR file, move the TLD file to the META-INF directory, move the class files to thetop-level directory, repackage the WAR, and place that WAR file in your webapp's WEB-INF directory.

Answer: C

Q 20. Which path is required to be present within a WAR file?

A. /classes
B. /index.html
C. /MANIFEST-INF
D. /WEB-INF/web.xml
E. /WEB-INF/classes
F. /WEB-INF/index.html
G. /META-INF/index.xml

Answer: D

Q21. Given:

11. <servlet>
12. <servlet-name>catalog</servlet-name>
13. <jsp-file>/catalogTemplate.jsp</jsp-file>
14. <load-on-startup>10</load-on-startup>
15. </servlet>
Which two are true? (Choose two.)

A. Line 13 is not valid for a servlet declaration.
B. Line 14 is not valid for a servlet declaration.
C. One instance of the servlet will be loaded at startup.
D. Ten instances of the servlet will be loaded at startup.
E. The servlet will be referenced by the name catalog in mappings.

Answer: C, E

Q22. You have built a web application with tight security. Several directories of your webapp are used for internal purposes and you have overridden the default servlet to send an HTTP 403 status code for any request that maps to one of these directories. During testing, the Quality Assurance director decided that they did NOT like seeing the bare response page generated by Firefox and Internet Explorer. The director recommended that the webapp should return a more user-friendly web page that has the same look-and-feel as the webapp plus links to the webapp's search engine. You have created this JSP page in the /WEB-INF/jsps/error403.jsp file. You do NOT want to alter the complex logic of the default servlet. How can you declare that the web container must send this JSP page whenever a 403 status is generated?

A. <error-page>
<error-code>403</error-code>
<url>/WEB-INF/jsps/error403.jsp</url>
</error-page>

B. <error-page>
<status-code>403</status-code>
<url>/WEB-INF/jsps/error403.jsp</url>
</error-page>
C. <error-page>
<error-code>403</error-code>
<location>/WEB-INF/jsps/error403.jsp</location>
</error-page>
D. <error-page>
<status-code>403</status-code>
<location>/WEB-INF/jsps/error403.jsp</location>
</error-page>
Answer: C

Q23. Given a portion of a valid Java EE web application's directory structure:

```
MyApp
|
|-- Directory1
|      |-- File1.html
|
|-- META-INF
|          |-- File2.html
|
|-- WEB-INF
       |-- File3.html
```

You want to know whether File1.html, File2.html, and/or File3.html is protected from direct access by your web client's browsers.
What statement is true?
A. All three files are directly accessible.
B. Only File1.html is directly accessible.
C. Only File2.html is directly accessible.
D. Only File3.html is directly accessible.
E. Only File1.html and File2.html are directly accessible.
F. Only File1.html and File3.html are directly accessible.
G. Only File2.html and File3.html are directly accessible.

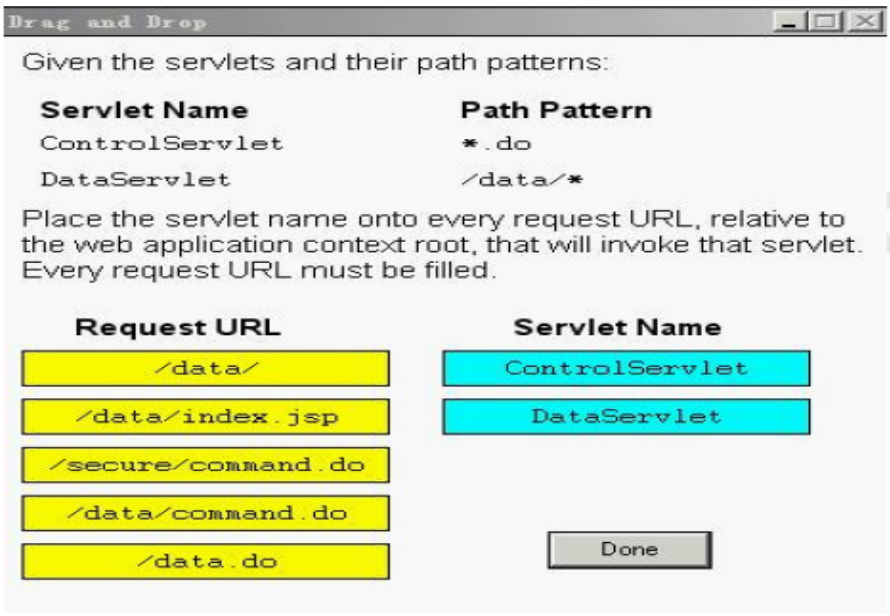
Answer: B

Q24. A web component accesses a local EJB session bean with a component interface of com.example.Account with a home interface of com.example.AccountHome and a JNDI reference of ejb/Account. Which makes the local EJB component accessible to the web components in the web application deployment descriptor?

A. <env-ref>
<ejb-ref-name>ejb/Account</ejb-ref-name>
<ejb-ref-type>Session</ejb-ref-type>
<local-home>com.example.AccountHome</local-home>

<local>com.example.Account</local>
</env-ref>
B. <resource-ref>
<ejb-ref-name>ejb/Account</ejb-ref-name>
<ejb-ref-type>Session</ejb-ref-type>
<local-home>com.example.AccountHome</local-home>
<local>com.example.Account</local>
</resource-ref>
C. <ejb-local-ref>
<ejb-ref-name>ejb/Account</ejb-ref-name>
<ejb-ref-type>Session</ejb-ref-type>
<local-home>com.example.AccountHome</local-home>
<local>com.example.Account</local>
</ejb-local-ref>
D. <ejb-remote-ref>
<ejb-ref-name>ejb/Account</ejb-ref-name>
<ejb-ref-type>Session</ejb-ref-type>
<local-home>com.example.AccountHome</local-home>
<local>com.example.Account</local>
</ejb-remote-ref>
Answer: C

Q 25. Click the Task button.
Place the servlet name onto every request URL, relative to the web application context root, that will invoke that servlet. Every request URL must be filled.



Answers:
/data/ --->DataServlet
/data/index.jsp --->DataServlet
/secure/command.do --->ControlServlet

/data/command.do --->DataServlet
/data.do --->ControlServlet

Q26. Given a portion of a valid Java EE web application's directory structure:

```
MyApp
|
|-- File1.html
|
|-- Directory1
|      |-- File2.html |
|-- META-INF
      |-- File3.html
```

You want to know whether File1.html, File2.html, and/or File3.html will be directly accessible by your web client's browsers.
Which statement is true?
A. All three files are directly accessible. B. Only File1.html is directly accessible.
C. Only File2.html is directly accessible. D. Only File3.html is directly accessible.
E. Only File1.html and File2.html are directly accessible.
F. Only File1.html and File3.html are directly accessible.
G. Only File2.html and File3.html are directly accessible.

Answer: E

Q 27. You have created a servlet that generates weather maps. The data for these maps is calculated by a remote host. The IP address of this host is usually stable, but occasionally does have to change as the corporate network grows and changes. This IP address used to be hard coded, but after the fifth change to the IP address in two years, you have decided that this value should be declared in the deployment descriptor so you do NOT have the recompile the web application every time the IP address changes. Which deployment descriptor snippet accomplishes this goal?

A. <serlvet-param>
<name>WeatherServlet.hostIP</name>
<value>127.0.4.20</value>
</servlet-param>
B. <init-param>
<name>WeatherServlet.hostIP</name>
<value>127.0.4.20</value>
</init-param>
C. <servlet>
<!-- servlet definition here -->
<param-name>WeatherServlet.hostIP</param-name>
<param-value>127.0.4.20</param-value>
</servlet>
D. <init-param>
<param-name>WeatherServlet.hostIP</param-name>
<param-value>127.0.4.20</param-value>

</init-param>
E. <serlvet-param>
<param-name>WeatherServlet.hostIP</param-name>
<param-value>127.0.4.20</param-value>
</servlet-param>
Answer: D

Q 28. In which two locations can library dependencies be defined for a web application? (Choose two.)
A. the web application deployment descriptor
B. the /META-INF/dependencies.xml file
C. the /META-INF/MANIFEST.MF manifest file
D. the /META-INF/MANIFEST.MF manifest of a JAR in the web application classpath
Answer: C, D

Q29. Which two about WAR files are true? (Choose two.)

A. WAR files must be located in the web application library directory.
B. WAR files must contain the web application deployment descriptor.
C. WAR files must be created by using archive tools designed specifically for that purpose.
D. The web container must serve the content of any META-INF directory located in a WAR file.
E. The web container must allow access to resources in JARs in the web application library directory.

Answer: B, E

Q30. Given this fragment from a Java EE deployment descriptor:

```
341. <error-page>
342. <exception-type>java.lang.Throwable</exception-type>
343. <location>/mainError.jsp</location>
344. </error-page>
345. <error-page>
346. <exception-type>java.lang.ClassCastException</exception-type>
347. <location>/castError.jsp</location>
348. </error-page>
```

If the web application associated with the fragment above throws a ClassCastException.

Which statement is true?
A. The deployment descriptor is invalid.
B. The container invokes mainError.jsp.
C. The container invokes castError.jsp.
D. Neither mainError.jsp nor castError.jsp is invoked.
Answer: C

Q31. Which defines the welcome files in a web application deployment descriptor?

A. <welcome>
<welcome-file>/welcome.jsp</welcome-file>

</welcome>
<welcome>
<welcome-file>/index.html</welcome-file>
</welcome>
B. <welcome-file-list>
<welcome-file>welcome.jsp</welcome-file>
<welcome-file>index.html</welcome-file>
</welcome-file-list>
C. <welcome>
<welcome-file>welcome.jsp</welcome-file>
</welcome>

<welcome>
<welcome-file>index.html</welcome-file>
</welcome>
D. <welcome-file-list>
<welcome-file>/welcome.jsp</welcome-file>
<welcome-file>/index.html</welcome-file>
</welcome-file-list>
E. <welcome>
<welcome-file>
www.TestsNow.com
- 120 -
<welcome-name>Welcome</welcome-name>
<location>welcome.jsp</location>
</welcome-file>
<welcome-file>
<welcome-name>Index</welcome-name>
<location>index.html</location>
</welcome-file>
</welcome>

Answer: B

Q32. In which three directories, relative to a web application's root, may a tag library descriptor file reside when deployed directly into a web application? (Choose three.)

A. /WEB-INF
B. /META-INF
C. /WEB-INF/tlds
D. /META-INF/tlds
E. /WEB-INF/resources
F. /META-INF/resources
Answer: A, C, E

Unit- 3: The Web Container Model

Objectives

1. For the ServletContext initialization parameters: write servlet code to access initialization parameters; and create the deployment descriptor elements for declaring initialization parameters.
2.For the fundamental servlet attribute scopes (request, session, and context): write servlet code to add, retrieve, and remove attributes; given a usage scenario, identify the proper scope for an attribute; and identify multi-threading issues associated with each scope.
3. Describe the Web container request processing model; write and configure a filter; create a request or response wrapper; and given a design problem, describe how to apply a filter or a wrapper.
4. Describe the Web container life cycle event model for requests, sessions, and web applications;create and configure listener classes for each scope life cycle; create and configure scope attribute listener classes; and given a scenario, identify the proper attribute listener to use.
5. Describe the RequestDispatcher mechanism; write servlet code to create a request dispatcher; write servlet code to forward or include the target resource; and identify and describe the additional request-scoped attributes provided by the container to the target resource.

Q1.Servlet A receives a request that it forwards to servlet B within another web application in the same web container. Servlet A needs to share data with servlet B and that data must not be visible to other servlets in A's web application. In which object can the data that A shares with B be stored?

A. HttpSession
B. ServletConfig
C. ServletContext
D. HttpServletRequest
E. HttpServletResponse

Answer: D

Q2.our web site has many user-customizable features, for example font and color preferences on web pages. Your IT department has already built a subsystem for user preferencessing the Java SE platform's lang.util.prefs package APIs, and you have been ordered to reuse this subsystem in your web application. You need to create an event listener that constructs the preferences actory and stores it in the application scope for later use. Furthermore, this factory requires that the URL to a database must be declared in the deployment descriptor like this:

```
42. <context-param>
43. <param-name>prefsDbURL</param-name>
44. <param-value>
45. jdbc:pointbase:server://dbhost:4747/prefsDB
46. </param-value>
47. </context-param>
```

Which partial listener class will accomplish this goal?

```
A. public class PrefsFactoryInitializer implements ContextListener {
public void contextInitialized(ServletContextEvent e) {
ServletContext ctx = e.getContext();
String prefsURL = ctx.getParameter("prefsDbURL");
PreferencesFactory myFactory = makeFactory(prefsURL);
ctx.putAttribute("myPrefsFactory", myFactory);
}
// more code here
}
B. public class PrefsFactoryInitializer implements ServletContextListener {
public void contextCreated(ServletContext ctx) {
String prefsURL = ctx.getInitParameter("prefsDbURL");
PreferencesFactory myFactory = makeFactory(prefsURL);
ctx.setAttribute("myPrefsFactory", myFactory);
}
// more code here
}
C. public class PrefsFactoryInitializer implements ServletContextListener {
public void contextInitialized(ServletContextEvent e) {
ServletContext ctx = e.getServletContext();
String prefsURL = ctx.getInitParameter("prefsDbURL");
PreferencesFactory myFactory = makeFactory(prefsURL);
ctx.setAttribute("myPrefsFactory", myFactory);
}
// more code here
}
D. public class PrefsFactoryInitializer implements ContextListener {
public void contextCreated(ServletContext ctx) {
String prefsURL = ctx.getParameter("prefsDbURL");
PreferencesFactory myFactory = makeFactory(prefsURL);
ctx.putAttribute("myPrefsFactory", myFactory);
}
// more code here
}
```

Answer: C

Q3.developer wants a web application to be notified when the application is about to be shut down. Which two actions are necessary to accomplish this goal? (Choose two.)

A. include a listener directive in a JSP page
B. configure a listener in the TLD file using the <listener> element
C. include a <servlet-destroy> element in the web application deployment descriptor
D. configure a listener in the application deployment descriptor, using the <listener> element
E. include a class implementing ServletContextListener as part of the web application deployment
F. include a class implementing ContextDestroyedListener as part of the web application deployment
G. include a class implementing HttpSessionAttributeListener as part of the web application deployment

Answer: D, E

Q4.You want to create a filter for your web application and your filter will implement javax.servlet.Filter.
Which two statements are true? (Choose two.)

A. Your filter class must implement an init method and a destroy method.
B. Your filter class must also implement javax.servlet.FilterChain.
C. When your filter chains to the next filter, it should pass the same arguments it received in its doFilter method.
D. The method that your filter invokes on the object it received that implements javax.servlet.FilterChain can invoke either another filter or a servlet.
E. Your filter class must implement a doFilter method that takes, among other things, an HTTPServletRequest object and an HTTPServletResponse object.
Answer: A, D

Q5.Which three are true about the HttpServletRequestWrapper class? (Choose three.)

A. The HttpServletRequestWrapper is an example of the Decorator pattern.
B. The HttpServletRequestWrapper can be used to extend the functionality of a servlet request.
C. A subclass of HttpServletRequestWrapper CANNOT modify the behavior of the getReader method.
D. An HttpServletRequestWrapper may be used only by a class implementing the javax.servlet.Filter interface.
E. An HttpServletRequestWrapper CANNOT be used on the request passed to the RequestDispatcher.include method.
F. An HttpServletRequestWrapper may modify the header of a request within an object implementing the javax.servlet.Filter interface.

Answer: A, B, F

Q6.A developer wants to make a name attribute available to all servlets associated with a particular user, across multiple requests from that user, from the same browser instance.
Which two provide this capability from within a tag handler? (Choose two.)
A. pageContext.setAttribute("name", theValue);
B. pageContext.setAttribute("name", getSession());
C. pageContext.getRequest().setAttribute("name", theValue);
D. pageContext.getSession().setAttribute("name", theValue);
E. pageContext.setAttribute("name", theValue, PageContext.PAGE_SCOPE);
F. pageContext.setAttribute("name", theValue, PageContext.SESSION_SCOPE);
Answer: D, F

Q7.Click the Exhibit button.

The resource requested by the RequestDispatcher is available and implemented by the DestinationServlet.
What is the result?

```
// From file SourceServlet.java
11. public class SourceServlet extends
HttpServlet {
12.   public void service(HttpServletRequest
request,
13.                       HttpServletResponse
response)
14.         throws ServletException,
IOException {
15.     ServletContext
cxt=getServletConfig().getServletContext();
16.     RequestDispatcher rd =
17.               cxt.getRequestDispatcher("/des
n");
18.     response.getWriter().println("hello
from source");
19.     response.flushBuffer();
20.     rd.forward(request, response);
21.   }
22. }

// From file DestinationServlet.java
11. public class DestinationServlet extends
HttpServlet {
12.   public void service(HttpServletRequest
request,
13.                       HttpServletResponse
response)
14.         throws ServletException,
IOException {
15.     response.getWriter().println("hello
from dest");
17.     response.flushBuffer();
18.   }
19. }
```

A. An exception is thrown at runtime by SourceServlet.
B. An exception is thrown at runtime by DestinationServlet.
C. Only "hello from dest" appears in the response output stream.
D. Both "hello from source" and "hello from dest" appear in the response output stream.
Answer: A

Q8.Given the definition of MyServlet:

```
11. public class MyServlet extends HttpServlet {
12. public void service(HttpServletRequest request,
13. HttpServletResponse response)
14. throws ServletException, IOException {
15. HttpSession session = request.getSession();
```

```
16 session.setAttribute("myAttribute","myAttributeValue");
17. session.invalidate();
18. response.getWriter().println("value=" +
19. session.getAttribute("myAttribute"));
20. }
21. }
```

What is the result when a request is sent to MyServlet?
A. An IllegalStateException is thrown at runtime.
B. An InvalidSessionException is thrown at runtime.
C. The string "value=null" appears in the response stream.
D. The string "value=myAttributeValue" appears in the response stream.

Answer: A

Q9.You need to store a Java long primitive attribute, called customerOID, into the session scope. Which two code snippets allow you to insert this value into the session? (Choose two.)

A. long customerOID = 47L;
session.setAttribute("customerOID", new Long(customerOID));
B. long customerOID = 47L;
session.setLongAttribute("customerOID", new Long(customerOID));
C. long customerOID = 47L;
session.setAttribute("customerOID", customerOID);
D. long customerOID = 47L;
session.setNumericAttribute("customerOID", new Long(customerOID));
E. long customerOID = 47L;
session.setLongAttribute("customerOID", customerOID);
F. long customerOID = 47L;
session.setNumericAttribute("customerOID", customerOID);

Answer: A, C

Q10.Your web application requires the adding and deleting of many session attributes during a complex use case. A bug report has come in that indicates that an important session attribute is being deleted too soon and a NullPointerException is being thrown several interactions after the fact. You have decided to create a session event listener that will log when attributes are being deleted so you can track down when the attribute is erroneously being deleted.

Which listener class will accomplish this debugging goal?

A. Create an HttpSessionAttributeListener class and implement the attributeDeleted method and log the attribute name using the getName method on the event object.

B. Create an HttpSessionAttributeListener class and implement the attributeRemoved method and log the attribute name using the getName method on the event object.

C. Create an SessionAttributeListener class and implement the attributeRemoved method and log the attribute name using the getAttributeName method on the event object.

D. Create an SessionAttributeListener class and implement the attributeDeleted method and log the attribute name using the getAttributeName method on the event object.
Answer: B

Q11.One of the use cases in your web application uses many session-scoped attributes. At the end of the use case, you want to clear out this set of attributes from the session object.
Assume that this static variable holds this set of attribute names:

```
201. private static final Set<String> USE_CASE_ATTRS;
202. static {
203. USE_CASE_ATTRS.add("customerOID");
204. USE_CASE_ATTRS.add("custMgrBean");
205. USE_CASE_ATTRS.add("orderOID");
206. USE_CASE_ATTRS.add("orderMgrBean");
207. }
```

Which code snippet deletes these attributes from the session object?

```
A. session.removeAll(USE_CASE_ATTRS);
B. for ( String attr : USE_CASE_ATTRS ) {
session.remove(attr);
}
C. for ( String attr : USE_CASE_ATTRS ) {
session.removeAttribute(attr);
}
D. for ( String attr : USE_CASE_ATTRS ) {
session.deleteAttribute(attr);
}
E. session.deleteAllAttributes(USE_CASE_ATTRS);
```

Answer: C

Q12. You have a simple web application that has a single Front Controller servlet hat dispatches to JSPs to generate a variety of views. Several of these views require further database processing to retrieve the necessary order object using the orderID request parameter. To do this additional processing, you pass the request first to a servlet that is mapped to the URL pattern WEB-INF/retreiveorder.do in the deployment descriptor. This servlet takes two request parameters, the orderID and the jspURL. It handles the database calls to retrieve and build the complex order objects and then it dispatches to the jspURL. Which code snippet in the Front Controller servlet dispatches the request to the order retrieval servlet?

```
A. request.setAttribute("orderID", orderID);
request.setAttribute("jspURL", jspURL);
RequestDispatcher view
= context.getRequestDispatcher("/WEB-INF/retreiveOrder.do");
view.forward(request, response);

B. request.setParameter("orderID", orderID);
request.setParameter("jspURL", jspURL);
Dispatcher view
```

= request.getDispatcher("/WEB-INF/retreiveOrder.do");
view.forwardRequest(request, response);

C. String T="/WEB-INF/retreiveOrder.do?orderID=%d&jspURL=%s";
String url = String.format(T, orderID, jspURL);
RequestDispatcher view
= context.getRequestDispatcher(url);
view.forward(request, response);

D. String T="/WEB-INF/retreiveOrder.do?orderID=%d&jspURL=%s";
String url = String.format(T, orderID, jspURL);
Dispatcher view = context.getDispatcher(url);
view.forwardRequest(request, response);
Answer: C

Q13.You want to create a filter for your web application and your filter will implement javax.servlet.Filter.
Which two statements are true? (Choose two.)
A. Your filter class must implement an init method and a destroy method.
B. Your filter class must also implement javax.servlet.FilterChain.
C. When your filter chains to the next filter, it should pass the same arguments it received in its doFilter method.
D. The method that your filter invokes on the object it received that implements javax.servlet.FilterChain can invoke either another filter or a servlet.
E. Your filter class must implement a doFilter method that takes, among other things, an HTTPServletRequest object and an HTTPServletResponse object.
Answer: A, D

Q14. Given the web application deployment descriptor elements:
11. <filter>
12. <filter-name>ParamAdder</filter-name>
13. <filter-class>com.example.ParamAdder</filter-class>
14. </filter>
...
24. <filter-mapping>
25. <filter-name>ParamAdder</filter-name>
26. <servlet-name>MyServlet</servlet-name>
27. <!-- insert element here -->
28. </filter-mapping>
Which element, inserted at line 27, causes the ParamAdder filter to be applied when MyServlet is invoked by another servlet using the RequestDispatcher.include method?

A. <include/>
B. <dispatcher>INCLUDE</dispatcher>
C. <dispatcher>include</dispatcher>
D. <filter-condition>INCLUDE</filter-condition>
E. <filter-condition>include</filter-condition>
Answer: B

Q15. Your web application uses a simple architecture in which servlets handle requests and then forward to a JSP using a request dispatcher. You need to pass information calculated by the servlet to the JSP; furthermore, that JSP uses a custom tag and must also process this information. This information must NOT be accessible to any other servlet, JSP or session in the webapp. How can you accomplish this goal?

A. Store the data in a public instance variable in the servlet.
B. Add an attribute to the request object before using the request dispatcher.
C. Add an attribute to the context object before using the request dispatcher.
D. This CANNOT be done as the tag handler has no means to extract this data.
Answer: B

Q16. A developer chooses to avoid using SingleThreadModel but wants to ensure that data is updated in a thread-safe manner. Which two can support this design goal? (Choose two.)
A. Store the data in a local variable.
B. Store the data in an instance variable.
C. Store the data in the HttpSession object.
D. Store the data in the ServletContext object.
E. Store the data in the ServletRequest object.
Answer: A, E

Q17. Your web application uses a simple architecture in which servlets handle requests and then forward to a JSP using a request dispatcher. You need to pass information calculated in the servlet to the JSP for view generation. This information must NOT be accessible to any other servlet, JSP or session in the webapp. Which two techniques can you use to accomplish this goal?(Choose two.)

A. Add attributes to the session object.
B. Add attributes on the request object.
C. Add parameters to the request object.
D. Use the pageContext object to add request attributes.
E. Add parameters to the JSP's URL when generating the request dispatcher.

Answer: B

Q18. Given:
String value = getServletContext().getInitParameter("foo");
in an HttpServlet and a web application deployment descriptor that contains:

```
<context-param>
<param-name>foo</param-name>
<param-value>frodo</param-value>
</context-param>
```

Which two are true? (Choose two.)

A. The foo initialization parameter CANNOT be set programmatically.
B. Compilation fails because getInitParameter returns type Object.
C. The foo initialization parameter is NOT a servlet initialization parameter.

D. Compilation fails because ServletContext does NOT have a getInitParameter method.
E. The foo parameter must be defined within the <servlet> element of the deployment descriptor.
F. The foo initialization parameter can also be retrieved using getServletConfig().getInitParameter("foo").

Answer: A, C

Q19. Click the Exhibit button. Given the web application deployment descriptor elements:

```
11. <filter>
12. <filter-name>ParamAdder</filter-name>
13. <filter-class>com.example.ParamAdder</filter-class>
14. </filter>
31. <filter-mapping>
32. <filter-name>ParamAdder</filter-name>
33. <servlet-name>Destination</servlet-name>
34. </filter-mapping>
...
55. <servlet-mapping>
56. <servlet-name>Destination</servlet-name>
57. <url-pattern>/dest/Destination</url-pattern>
58. </servlet-mapping>
```

What is the result of a client request of the Source servlet with no query string?

```
// Source Servlet : Source.java
10.  public class Source extends HttpServlet {
11.    public void service(HttpServletRequest request,
12.                        HttpServletResponse response)
13.                 throws ServletException, IOException {
14.      RequestDispatcher rd =
15.        request.getRequestDispatcher("/dest/Destination");
16.      rd.forward(request, response);
17.    }
18.  }

// Filter : ParamAdder.java
12.  public class ParamAdder implements Filter {
       // ...
23.    public void doFilter(ServletRequest request,
24.                         ServletResponse response,
25.                         FilterChain chain)
26.           throws ServletException, IOException {
27.      request.setAttribute("filterAdded", "addedByFilter");
28.      chain.doFilter(request, response);
29.    }
       // ...
50.  }

//  Destination Servlet Destination.java
10.  public class Destination extends HttpServlet {
11.    public void service(HttpServletRequest request,
12.                        HttpServletResponse response)
13.                 throws ServletException, IOException {
14.      String filterParam =
15.         (String) request.getAttribute("filterAdded");
16.      response.getWriter().println("filterAdded = "
17.                                   + filterParam);
18.    }
19.  }
```

A. The output "filterAdded = null" is written to the response stream.
B. The output "filterAdded = addedByFilter" is written to the response stream.
C. An exception is thrown at runtime within the service method of the Source servlet.
D. An exception is thrown at runtime within the service method of the Destination servlet.
Answer: A

Q20.Given a Filter class definition with this method:

```
21. public void doFilter(ServletRequest request,
22. ServletResponse response,
23. FilterChain chain)
24. throws ServletException, IOException {
25. // insert code here
26. }
```

Which should you insert at line 25 to properly invoke the next filter in the chain, or the target servlet if there are no more filters?

A. chain.forward(request, response);
B. chain.doFilter(request, response);
C. request.forward(request, response);
D. request.doFilter(request, response);

Answer: B

Q21. Servlet A forwarded a request to servlet B using the forward method of RequestDispatcher. What attribute in B's request object contains the URI of the original request received by servlet A?

A. REQUEST_URI
B. javax.servlet.forward.request_uri
C. javax.servlet.forward.REQUEST_URI
D. javax.servlet.request_dispatcher.request_uri
E. javax.servlet.request_dispatcher.REQUEST_URI
Answer: B

Q22.One of the use cases in your web application uses many session-scoped attributes. At the end of the use case, you want to clear out this set of attributes from the session object. Assume that this static variable holds this set of attribute names:

```
201. private static final Set<String> USE_CASE_ATTRS;
202. static {
203. USE_CASE_ATTRS.add("customerOID");
204. USE_CASE_ATTRS.add("custMgrBean");
205. USE_CASE_ATTRS.add("orderOID");
206. USE_CASE_ATTRS.add("orderMgrBean");
207. }
```

Which code snippet deletes these attributes from the session object?
A. session.removeAll(USE_CASE_ATTRS);

B. for (String attr : USE_CASE_ATTRS) {
session.remove(attr);
}
C. for (String attr : USE_CASE_ATTRS) {
session.removeAttribute(attr);
}
D. for (String attr : USE_CASE_ATTRS) {
session.deleteAttribute(attr); }
E. session.deleteAllAttributes(USE_CASE_ATTRS);
Answer: C

Q23. You need to store a floating point number, called Tsquare, in the session scope. Which two code snippets allow you to retrieve this value? (Choose two.)

A. float Tsquare = session.getFloatAttribute("Tsquare");
B. float Tsquare = (Float) session.getAttribute("Tsquare");
C. float Tsquare = (float) session.getNumericAttribute("Tsquare");
D. float Tsquare = ((Float) session.getAttribute.("Tsquare")).floatValue();
E. float Tsquare = ((Float) session.getFloatAttribute.("Tsquare")).floatValue;
F. float Tsquare = ((Float) session.getNumericAttribute.("Tsquare")).floatValue;

Answer: B, D

Q24. You need to store a floating point number, called Tsquare, in the session scope. Which two code snippets allow you to retrieve this value? (Choose two.)

A. float Tsquare = session.getFloatAttribute("Tsquare");
B. float Tsquare = (Float) session.getAttribute("Tsquare");
C. float Tsquare = (float) session.getNumericAttribute("Tsquare");
D. float Tsquare = ((Float) session.getAttribute.("Tsquare")).floatValue();
E. float Tsquare = ((Float) session.getFloatAttribute.("Tsquare")).floatValue;
F. float Tsquare = ((Float) session.getNumericAttribute.("Tsquare")).floatValue;

Answer: B, D

Q25.You need to store a Java long primitive attribute, called customerOID, into the session scope. Which two code snippets allow you to insert this value into the session? (Choose two.)

A. long customerOID = 47L;
session.setAttribute("customerOID", new Long(customerOID));
B. long customerOID = 47L;
session.setLongAttribute("customerOID", new Long(customerOID));
C. long customerOID = 47L;
session.setAttribute("customerOID", customerOID);
D. long customerOID = 47L;
session.setNumericAttribute("customerOID", new Long(customerOID));
E. long customerOID = 47L;
session.setLongAttribute("customerOID", customerOID);
F. long customerOID = 47L;

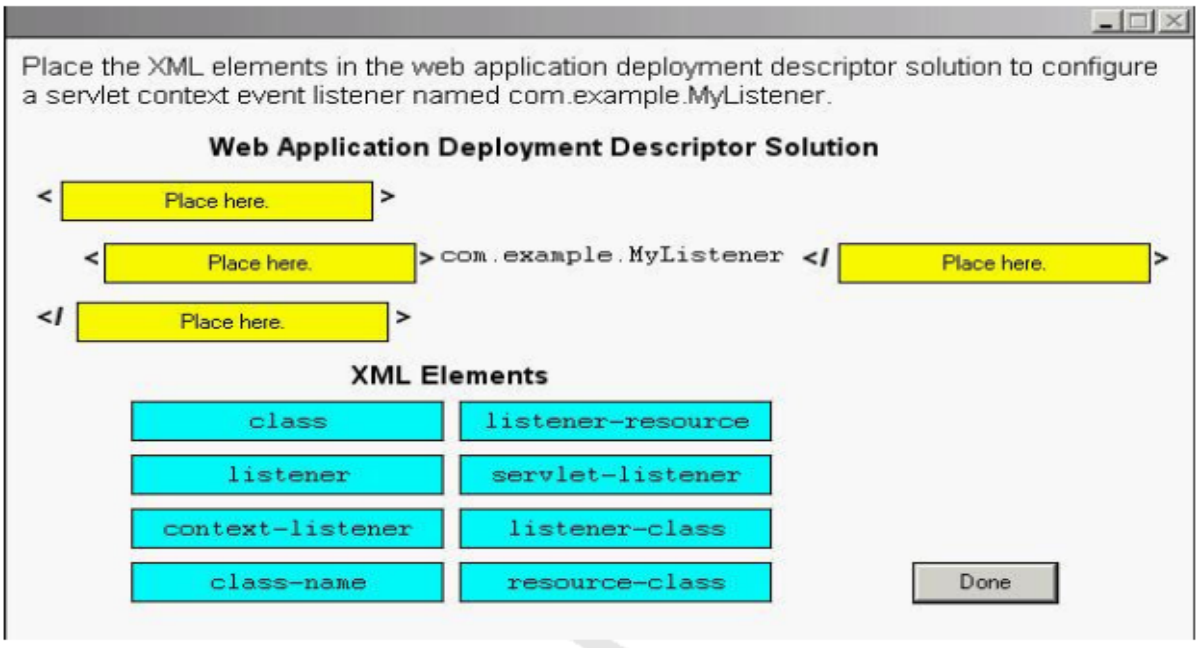
session.setNumericAttribute("customerOID", customerOID);

Answer: A, C

Q26. Your web application uses a simple architecture in which servlets handle requests and then forward to a JSP using a request dispatcher. You need to pass information calculated in the servlet to the JSP for view generation. This information must NOT be accessible to any other servlet, JSP or session in the webapp. Which two techniques can you use to accomplish this goal? (Choose two.)
A. Add attributes to the session object. B. Add attributes on the request object.
C. Add parameters to the request object.
D. Use the pageContext object to add request attributes.
E. Add parameters to the JSP's URL when generating the request dispatcher.
Answer: B, E

Q 27. Which three are true about servlet filters? (Choose three.)
A. A filter must implement the destroy method.
B. A filter must implement the doFilter method.
C. A servlet may have multiple filters associated with it.
D. A servlet that is to have a filter applied to it must implement the javax.servlet. FilterChain interface.
E. A filter that is part of a filter chain passes control to the next filter in the chain by invoking the FilterChain.forward method.
F. For each <filter> element in the web application deployment descriptor, multiple instances of a filter may be created by the web container.
Answer: A, B, C

Q 28. Click the Task button.
Place the XML elements in the web application deployment descriptor solution to configure a servlet context event listener named com.example.MyListener.



Answers:
<listener>
<listener-class>com.example.MyListener</listener-class>
</listener>

Q 29. Which is true about the web container request processing model?

A. The init method on a filter is called the first time a servlet mapped to that filter is invoked.
B. A filter defined for a servlet must always forward control to the next resource in the filter chain.
C. Filters associated with a named servlet are applied in the order they appear in the web application deployment descriptor file.
D. If the init method on a filter throws an UnavailableException, then the container will make no further attempt to execute it.

Answer: C

Q30. Your IT department is building a lightweight Front Controller servlet that invokes an application logic object with the interface:

public interface ApplicationController {
public String invoke(HttpServletRequest request)
}

The return value of this method indicates a symbolic name of the next view. From this name, the Front Controller servlet looks up the JSP URL in a configuration table. This URL might be an absolute path or a path relative to the current request. Next, the Front Controller servlet must send the request to this JSP to generate the view. Assume that the servlet variable request is assigned the current HttpServletRequest object and the variable context is assigned the webapp's ServletContext.
Which code snippet of the Front Controller servlet accomplishes this goal?

A. Dispatcher view
= context.getDispatcher(viewURL);
view.forwardRequest(request, response);

B. Dispatcher view
= request.getDispatcher(viewURL);
view.forwardRequest(request, response);

C. RequestDispatcher view
= context.getRequestDispatcher(viewURL);
view.forward(request, response);

D. RequestDispatcher view = request.getRequestDispatcher(viewURL);
view.forward(request, response);
Answer: D

Q31. Given that a web application consists of two HttpServlet classes, ServletA and ServletB, and the ServletA.service method:
20. String key = "com.example.data";
21. session.setAttribute(key, "Hello");
22. Object value = session.getAttribute(key);
23.
Assume session is an HttpSession, and is not referenced anywhere else in ServletA.

Which two changes, taken together, ensure that value is equal to "Hello" on line 23? (Choose two.)
A. ensure that the ServletB.service method is synchronized
B. ensure that the ServletA.service method is synchronized
C. ensure that ServletB synchronizes on the session object when setting session attributes
D. enclose lines 21-22 in a synchronized block:
synchronized(this) {
session.setAttribute(key, "Hello");
value = session.getAttribute(key);
}
E. enclose lines 21-22 in a synchronized block:
synchronized(session) {
session.setAttribute(key, "Hello");
value = session.getAttribute(key); }
Answer: C, E

Unit-4: Session Management

Objectives

1) Write servlet code to store objects into a session object and retrieve objects from a session object.
2) Given a scenario describe the APIs used to access the session object, explain when the session object was created, and describe the mechanisms used to destroy the session object, and when it was destroyed.
3) Using session listeners, write code to respond to an event when an object is added to a session, and write code to respond to an event when a session object migrates from one VM to another.
4) Given a scenario, describe which session management mechanism the Web container could employ, how cookies might be used to manage sessions, how URL rewriting might be used to manage sessions, and write servlet code to perform URL rewriting.

Q1. A developer for the company web site has been told that users may turn off cookie support in their browsers. What must the developer do to ensure that these customers can still use the web application?

A. The developer must ensure that every URL is properly encoded using the appropriate URL rewriting APIs.
B. The developer must provide an alternate mechanism for managing sessions and abandon the HttpSession mechanism entirely.
C. The developer can ignore this issue. Web containers are required to support automatic URL rewriting when cookies are not supported.
D. The developer must add the string id=<sessionid> to the end of every URL to ensure that the conversation with the browser can continue.

Answer: A

Q2. As a convenience feature, your web pages include an Ajax request every five minutes to a special servlet that monitors the age of the user's session. The client-side JavaScript that handles the Ajax callback displays a message on the screen as the session ages. The Ajax call does NOT pass any cookies, but it passes the session ID in a request parameter called sessionID. In addition, assume that your webapp keeps a hashmap of session objects by the ID. Here is a partial implementation of this servlet:

```
10. public class SessionAgeServlet extends HttpServlet {
11. public void service(HttpServletRequest request, HttpServletResponse) throws IOException {
12. String sessionID = request.getParameter("sessionID");
13. HttpSession session = getSession(sessionID);
14. long age = // your code here
15. response.getWriter().print(age);
16. } ... // more code here
47. }
```

Which code snippet on line 14, will determine the age of the session?

A. session.getMaxInactiveInterval();
B. session.getLastAccessed().getTime() - session.getCreationTime().getTime();
C. session.getLastAccessedTime().getTime() - session.getCreationTime().getTime();
D. session.getLastAccessed() - session.getCreationTime();
E. session.getMaxInactiveInterval() - session.getCreationTime();
F. session.getLastAccessedTime() - session.getCreationTime();

Answer: F

Q3.Which statement is true about web container session management?

A. Access to session-scoped attributes is guaranteed to be thread-safe by the web container.
B. To activate URL rewriting, the developer must use the HttpServletResponse. setURLRewriting method.
C. If the web application uses HTTPS, then the web container may use the data on the HTTPS request stream to identify the client.
D. The JSESSIONID cookie is stored permanently on the client so that a user may return to the web application and the web container will rejoin that session.
Answer: C

Q4. Your company has a corporate policy that prohibits storing a customer's credit card number in any corporate database. However, users have complained that they do NOT want to re-enter their credit card number for each transaction. Your management has decided to use client-side cookies to record the user's credit card number for 120 days. Furthermore, they also want to protect this information during transit from the web browser to the web container; so the cookie must only be transmitted over HTTPS. Which code snippet creates the "creditCard" cookie and adds it to the out going response to be stored on the user's web browser?

A.10. Cookie c = new Cookie("creditCard", usersCard);
11. c.setSecure(true);
12. c.setAge(10368000);
13. response.addCookie(c);

B. 10. Cookie c = new Cookie("creditCard", usersCard);
11. c.setHttps(true);
12. c.setMaxAge(10368000);
13. response.setCookie(c);

C. 10. Cookie c = new Cookie("creditCard", usersCard);
11. c.setSecure(true);
12. c.setMaxAge(10368000);
13. response.addCookie(c);

D. 10. Cookie c = new Cookie("creditCard", usersCard);
11. c.setHttps(true);
12. c.setAge(10368000);

13. response.addCookie(c);

E. 10. Cookie c = new Cookie("creditCard", usersCard);
11. c.setSecure(true);
12. c.setAge(10368000);
13. response.setCookie(c);
Answer: C

Q5. You need to retrieve the username cookie from an HTTP request. If this cookie does NOT exist, then the c variable will be null. Which code snippet must be used to retrieve this cookie object?

A. 10. Cookie c = request.getCookie("username");

B. 10. Cookie c = null;
11. for (Iterator i = request.getCookies();
12. i.hasNext();) {
13. Cookie o = (Cookie) i.next();
14. if (o.getName().equals("username")) {
15. c = o;
16. break;
17. }
18. }

C. 10. Cookie c = null;
11. for (Enumeration e = request.getCookies();
12. e.hasMoreElements();) {
13. Cookie o = (Cookie) e.nextElement();
14. if (o.getName().equals("username")) {
15. c = o;
16. break;
17. }
18. }

D. 10. Cookie c = null;
11. Cookie[] cookies = request.getCookies();
12. for (int i = 0; i < cookies.length; i++) {
13. if (cookies[i].getName().equals("username")) {
14. c = cookies[i];
15. break;
16. }
17. }

Answer: D

Q6. What is the purpose of session management?
A. To manage the user's login and logout activities.
B. To store information on the client-side between HTTP requests.
C. To store information on the server-side between HTTP requests.

D. To tell the web container to keep the HTTP connection alive so it can make subsequent requests without the
delay of making the TCP connection.

Answer: C

Q7. The Squeaky Beans Inc. shopping application was initially developed for a non-distributed environment. The company recently purchased the Acme Application Server, which supports distributed HttpSession objects. When deploying the application to the server, the deployer marks it as distributable in the web application deployment descriptor to take advantage of this feature.

Given this scenario, which two must be true? (Choose two.)

A. The J2EE web container must support migration of objects that implement Serializable.
B. The J2EE web container must use the native JVM Serialization mechanism for distributing HttpSession objects.
C. As per the specification, the J2EE web container ensures that distributed HttpSession objects will be stored in a database.
D. Storing references to Enterprise JavaBeans components in the HttpSession object might NOT be supported by J2EE web containers.

Answer: A, D

Q8. In your web application, you need to execute a block of code whenever the session object is first created. Which design will accomplish this goal?

A. Create an HttpSessionListener class and implement the sessionInitialized method with that block of code.
B. Create an HttpSessionActivationListener class and implement the sessionCreated method with that block of code.
C. Create a Filter class, call the getSession(false) method, and if the result was null, then execute that block of code.
D. Create an HttpSessionListener class and implement the sessionCreated method with that block of code.
E. Create a Filter class, call the getSession(true) method, and if the result was NOT null, then execute that block of code.

Answer: D

Q9. Which interface must a class implement so that instances of the class are notified after any object is added to a session?

A. javax.servlet.http.HttpSessionListener
B. javax.servlet.http.HttpSessionValueListener
C. javax.servlet.http.HttpSessionBindingListener
D. javax.servlet.http.HttpSessionAttributeListener

Answer: D

Q10. Which method must be used to encode a URL passed as an argument to HttpServletResponse.sendRedirect when using URL rewriting for session tracking?

A. ServletResponse.encodeURL
B. HttpServletResponse.encodeURL
C. ServletResponse.encodeRedirectURL
D. HttpServletResponse.encodeRedirectURL

Answer: D

Q11. Users of your web application have requested that they should be able to set the duration of their sessions. So for example, one user might want a webapp to stay connected for an hour rather than the webapp's default of fifteen minutes; another user might want to stay connected for a whole day. Furthermore, you have a special login servlet that performs user authentication and retrieves the User object from the database. You want to augment this code to set up the user's specified session duration.
Which code snippet in the login servlet will accomplish this goal?

A. User user = // retrieve the User object from the database
session.setDurationInterval(user.getSessionDuration());
B. User user = // retrieve the User object from the database
session.setMaxDuration(user.getSessionDuration());
C. User user = // retrieve the User object from the database
session.setInactiveInterval(user.getSessionDuration());
D. User user = // retrieve the User object from the database
session.setDuration(user.getSessionDuration());
E. User user = // retrieve the User object from the database
session.setMaxInactiveInterval(user.getSessionDuration());
F. User user = // retrieve the User object from the database
session.setMaxDurationInterval(user.getSessionDuration());

Answer: E

Q12. Which two classes or interfaces provide a getSession method? (Choose two.)

A. javax.servlet.http.HttpServletRequest
B. javax.servlet.http.HttpSessionContext
C. javax.servlet.http.HttpServletResponse
D. javax.servlet.http.HttpSessionBindingEvent
E. javax.servlet.http.HttpSessionAttributeEvent

Answer: A, D

Q13.You have built a web application that you license to small businesses. The webapp uses a context parameter, called licenseExtension, which enables certain advanced features based on your client's license package. When a client pays for a specific service, you provide them with a license extension key that they insert into the <context-param> of the deployment descriptor. Not every client will have this context parameter so you need to create a context listener to set up a default value in the licenseExtension parameter. Which code snippet will accomplish this goal?

A. You cannot do this because context parameters CANNOT be altered programmatically.
B. String ext = context.getParameter('licenseExtension');
if (ext == null) {
context.setParameter('licenseExtension', DEFAULT);
}
C. String ext = context.getAttribute('licenseExtension');
if (ext == null) {
context.setAttribute('licenseExtension', DEFAULT);
}
D. String ext = context.getInitParameter('licenseExtension');
if (ext == null) {
context.resetInitParameter('licenseExtension', DEFAULT);
}
E. String ext = context.getInitParameter('licenseExtension');
if (ext == null) {
context.setInitParameter('licenseExtension', DEFAULT);
}
Answer: A

Q14. You have a use case in your web application that adds several session-scoped attributes. At the end of the use case, one of these objects, the manager attribute, is removed and then it needs to decide which of the other session-scoped attributes to remove. How can this goal be accomplished?

A. The object of the manager attribute should implement the HttpSessionBindingListener and it should call the removeAttribute method on the appropriate session attributes.

B. The object of the manager attribute should implement the HttpSessionListener and it should call the removeAttribute method on the appropriate session attributes.

C. The object of the manager attribute should implement the HttpSessionBindingListener and it should call the deleteAttribute method on the appropriate session attributes.

D. The object of the manager attribute should implement the HttpSessionListener and it should call the deleteAttribute method on the appropriate session attributes.

Answer: A

Q15. Your web site has many user-customizable features, for example font and color preferences on web pages. Your IT department has already built a subsystem for user preferences using Java SE's lang.util.prefs package APIs and you have been ordered to reuse this subsystem in your web application. You need to create an event listener that stores the user's Preference object when an HTTP session is created. Also, note that user identification information is stored in an HTTP cookie.

Which partial listener class can accomplish this goal?

A. public class UserPrefLoader implements HttpSessionListener {
public void sessionCreated(HttpSessionEvent se) {

```
MyPrefsFactory myFactory = (MyPrefsFactory)
se.getServletContext().getAttribute("myPrefsFactory");
User user = getUserFromCookie(se);
myFactory.setThreadLocalUser(user);
Preferences userPrefs = myFactory.userRoot();
se.getSession().setAttribute("prefs", userPrefs);
}
// more code here
}
B. public class UserPrefLoader implements SessionListener {
public void sessionCreated(SessionEvent se) {
MyPrefsFactory myFactory = (MyPrefsFactory) se.getContext().getAttribute("myPrefsFactory");
User user = getUserFromCookie(se);
myFactory.setThreadLocalUser(user);
Preferences userPrefs = myFactory.userRoot();
se.getSession().addAttribute("prefs", userPrefs);
}
// more code here
}
C. public class UserPrefLoader implements HttpSessionListener {
public void sessionInitialized(HttpSessionEvent se) {
MyPrefsFactory myFactory = (MyPrefsFactory)
se.getServletContext().getAttribute("myPrefsFactory");
User user = getUserFromCookie(se);
myFactory.setThreadLocalUser(user);
Preferences userPrefs = myFactory.userRoot();
se.getHttpSession().setAttribute("prefs", userPrefs);
}
// more code here
}
D. public class UserPrefLoader implements SessionListener {
public void sessionInitialized(SessionEvent se) {
MyPrefsFactory myFactory = (MyPrefsFactory)
se.getServletContext().getAttribute("myPrefsFactory");
User user = getUserFromCookie(se);
myFactory.setThreadLocalUser(user);
Preferences userPrefs = myFactory.userRoot();
se.getSession().addAttribute("prefs", userPrefs);
}
// more code here
}
```

Answer: A

Q16. For which three events can web application event listeners be registered?(Choose three.)
A. when a session is created B. after a servlet is destroyed
C. when a session has timed out D. when a cookie has been created
E. when a servlet has forwarded a request F. when a session attribute value is changed
Answer: A, C, F

Q17. Given an HttpServletRequest request:

22. String id = request.getParameter("jsessionid");
23. // insert code here
24. String name = (String) session.getAttribute("name");
Which three can be placed at line 23 to retrieve an existing HttpSession object? (Choose three.)

A. HttpSession session = request.getSession();
B. HttpSession session = request.getSession(id);
C. HttpSession session = request.getSession(true);
D. HttpSession session = request.getSession(false);
E. HttpSession session = request.getSession("jsessionid");

Answer: A, C, D

Q18. A developer for the company web site has been told that users may turn off cookie support in their browsers. What must the developer do to ensure that these customers can still use the web application?

A. The developer must ensure that every URL is properly encoded using the appropriate URL rewriting APIs.
B. The developer must provide an alternate mechanism for managing sessions and abandon the HttpSession mechanism entirely.
C. The developer can ignore this issue. Web containers are required to support automatic URL rewriting when cookies are not supported.
D. The developer must add the string ?id=<sessionid> to the end of every URL to ensure that the conversation with the browser can continue.
Answer: A

Q19. Given the definition of MyObject and that an instance of MyObject is bound as a session attribute:
8. package com.example;
9. public class MyObject implements
10. javax.servlet.http.HttpSessionBindingListener {
11. // class body code here
12. }

Which is true?
A. Only a single instance of MyObject may exist within a session.
B. The unbound method of the MyObject instance is called when the session to which it is bound times out.
C. The com.example.MyObject must be declared as a servlet event listener in the web application deployment descriptor.
D. The valueUnbound method of the MyObject instance is called when the session to which it is bound times out.
Answer: D

Q 20. As a convenience feature, your web pages include an Ajax request every five minutes to a special servlet that monitors the age of the user's session. The client-side JavaScript that handles the Ajax callback displays a message on the screen as the session ages. The Ajax call does NOT pass any cookies, but it passes the session ID in a request parameter called sessionID. In addition, assume that your webapp keeps a hashmap of session objects by the ID. Here is a partial implementation of this servlet:

```
10. public class SessionAgeServlet extends HttpServlet {
11. public void service(HttpServletRequest request, HttpServletResponse) throws IOException {
12. String sessionID = request.getParameter("sessionID");
13. HttpSession session = getSession(sessionID);
14. long age = // your code here
15. response.getWriter().print(age);
16. }
... // more code here
47. }
```

Which code snippet on line 14, will determine the age of the session?
A. session.getMaxInactiveInterval();
B. session.getLastAccessed().getTime() - session.getCreationTime().getTime();
C. session.getLastAccessedTime().getTime() - session.getCreationTime().getTime();
D. session.getLastAccessed() - session.getCreationTime();
E. session.getMaxInactiveInterval() - session.getCreationTime();
F. session.getLastAccessedTime() - session.getCreationTime();

Answer: F

Q21. Which statement is true about web container session management?
A. Access to session-scoped attributes is guaranteed to be thread-safe by the web container.
B. To activate URL rewriting, the developer must use the HttpServletResponse. setURLRewriting method.
C. If the web application uses HTTPS, then the web container may use the data on the HTTPS request stream to identify the client.
D. The JSESSIONID cookie is stored permanently on the client so that a user may return to the web application and the web container will rejoin that session.

Answer: C

Q22. Given an HttpServletRequest request and an HttpServletResponse response:

```
41. HttpSession session = null;
42. // insert code here
43. if(session == null) {
44. // do something if session does not exist
45. } else {
46. // do something if session exists
47. }
```

To implement the design intent, which statement must be inserted at line 42?
A. session = response.getSession();
B. session = request.getSession();

C. session = request.getSession(true);
D. session = request.getSession(false);
E. session = request.getSession("jsessionid");
Answer: D

Q 23. A web application uses the HttpSession mechanism to determine if a user is "logged in." When a user supplies a valid user name and password, an HttpSession is created for that user.The user has access to the application for only 15 minutes after logging in. The code must determine how long the user has been logged in, and if this time is greater than 15 minutes, must destroy the HttpSession.

Which method in HttpSession is used to accomplish this?
A. getCreationTime B. invalidateAfter
C. getLastAccessedTime D. getMaxInactiveInterval
Answer: A

Q 24. Which method must be used to encode a URL passed as an argument to HttpServletResponse.sendRedirect when using URL rewriting for session tracking?
A. ServletResponse.encodeURL
B. HttpServletResponse.encodeURL
C. ServletResponse.encodeRedirectURL
D. HttpServletResponse.encodeRedirectURL
Answer: D

Q25.Which interface must a session attribute implement if it needs to be notified when a web container persists a session?
A. javax.servlet.http.HttpSessionListener
B. javax.servlet.http.HttpSessionBindingListener
C. javax.servlet.http.HttpSessionAttributeListener
D. javax.servlet.http.HttpSessionActivationListener

Answer: D

Q26.What is the purpose of session management?
A. To manage the user's login and logout activities.
B. To store information on the client-side between HTTP requests.
C. To store information on the server-side between HTTP requests.
D. To tell the web container to keep the HTTP connection alive so it can make subsequent requests without the delay of making the TCP connection.
Answer: C

Q27. Your company has a corporate policy that prohibits storing a customer's credit card number in any corporate database. However, users have complained that they do NOT want to re-enter their credit card number for each transaction. Your management has decided to use client-side cookies to record the user's credit card number for 120 days. Furthermore, they also want to protect this information during transit from the web browser to the web container; so the cookie must only be transmitted over HTTPS. Which code snippet creates the "creditCard" cookie and adds it to the out going response to be stored on the user's web browser?
A. 10. Cookie c = new Cookie("creditCard", usersCard);

11. c.setSecure(true);
12. c.setAge(10368000);
13. response.addCookie(c);

B. 10. Cookie c = new Cookie("creditCard", usersCard);
11. c.setHttps(true);
12. c.setMaxAge(10368000);
13. response.setCookie(c);

C. 10. Cookie c = new Cookie("creditCard", usersCard);
11. c.setSecure(true);
12. c.setMaxAge(10368000);
13. response.addCookie(c);

D. 10. Cookie c = new Cookie("creditCard", usersCard);
11. c.setHttps(true);
12. c.setAge(10368000);
13. response.addCookie(c);

E. 10. Cookie c = new Cookie("creditCard", usersCard);
11. c.setSecure(true);
12. c.setAge(10368000);
13. response.setCookie(c);

Answer: C

Unit-5 : Web Application Security

Objectives

1. Based on the servlet specification, compare and contrast the following security mechanisms: (a) authentication, (b) authorization, (c) data integrity, and (d) confidentiality.
2. In the deployment descriptor, declare a security constraint, a Web resource, the transport guarantee, the login configuration, and a security role.
3. Compare and contrast the authentication types (BASIC, DIGEST, FORM, and CLIENT-CERT); describe how the type works; and given a scenario, select an appropriate type.

Q1. Given:

```
3. class MyServlet extends HttpServlet {
4. public void doPut(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
5. // servlet code here ...
26. }
27. }
```

If the DD contains a single security constraint associated with MyServlet and its only <http-method> tags
and <auth-constraint> tags are:
<http-method>GET</http-method>
<http-method>PUT</http-method>
<auth-constraint>Admin</auth-constraint>

Which four requests would be allowed by the container? (Choose four.)

A. A user whose role is Admin can perform a PUT.
B. A user whose role is Admin can perform a GET.
C. A user whose role is Admin can perform a POST.
D. A user whose role is Member can perform a PUT.
E. A user whose role is Member can perform a POST.
F. A user whose role is Member can perform a GET.

Answer: A, B, C, E
Q2. What is true about Java EE authentication mechanisms?

a) If your deployment descriptor correctly declares an authentication type of CLIENT_CERT, your users must have a certificate from an official source before they can use your application.

b) If your deployment descriptor correctly declares an authentication type of BASIC, the container automatically requests a user name and password whenever a user starts a new session.

c) If you want your web application to support the widest possible array of browsers, and you want to perform authentication, the best choice of Java EE authentication mechanisms is DIGEST.

d) To use Java EE FORM authentication, you must declare two HTML files in your deployment descriptor, and you must use a predefined action in the HTML file that handles your user's login.

Answer: D

Q3. If you want to use the Java EE platform's built-in type of authentication that uses a custom HTML page for authentication, which two statements are true? (Choose two.)

A. Your deployment descriptor will need to contain this tag:
<auth-method>CUSTOM</auth-method>.
B. The related custom HTML login page must be named loginPage.html.
C. When you use this type of authentication, SSL is turned on automatically.
D. You must have a tag in your deployment descriptor that allows you to point to both a login HTML page and an HTML page for handling any login errors.
E. In the HTML related to authentication for this application, you must use predefined variable names for the variables that store the user and password values.
Answer: D, E

Q4. Given this fragment in a servlet:
23. if(req.isUserInRole("Admin")) {
24. // do stuff
25. }
And the following fragment from the related Java EE deployment descriptor:
812. <security-role-ref>
813. <role-name>Admin</role-name>
814. <role-link>Administrator</role-link>
815. </security-role-ref>
900. <security-role>
901. <role-name>Admin</role-name>
902. <role-name>Administrator</role-name>
903. </security-role>
What is the result?

A. Line 24 can never be reached.
B. The deployment descriptor is NOT valid.
C. If line 24 executes, the user's role will be Admin.
D. If line 24 executes, the user's role will be Administrator.
E. If line 24 executes the user's role will NOT be predictable.
Answer: D

Q5. Given the security constraint in a DD:

101. <security-constraint>
102. <web-resource-collection>
103. <web-resource-name>Foo</web-resource-name>

104. <url-pattern>/Bar/Baz/*</url-pattern>
105. <http-method>POST</http-method>
106. </web-resource-collection>
107. <auth-constraint>
108. <role-name>DEVELOPER</role-name>
109. </auth-constraint>
110. </security-constraint>

And given that "MANAGER" is a valid role-name, which four are true for this security constraint?(Choose four.)

A. MANAGER can do a GET on resources in the /Bar/Baz directory.
B. MANAGER can do a POST on any resource in the /Bar/Baz directory.
C. MANAGER can do a TRACE on any resource in the /Bar/Baz directory.
D. DEVELOPER can do a GET on resources in the /Bar/Baz directory.
E. DEVELOPER can do only a POST on resources in the /Bar/Baz directory.
F. DEVELOPER can do a TRACE on any resource in the /Bar/Baz directory.

Answer: A, C, D, F

Q6. Given the security constraint in a DD:

101. <security-constraint>
102. <web-resource-collection>
103. <web-resource-name>Foo</web-resource-name>
104. <url-pattern>/Bar/Baz/*</url-pattern>
105. <http-method>POST</http-method>
106. </web-resource-collection>
107. <auth-constraint>
108. <role-name>DEVELOPER</role-name>
109. </auth-constraint>
110. </security-constraint>
And given that "MANAGER" is a valid role-name, which four are true for this security constraint?(Choose four.)

A. MANAGER can do a GET on resources in the /Bar/Baz directory.
B. MANAGER can do a POST on any resource in the /Bar/Baz directory.
C. MANAGER can do a TRACE on any resource in the /Bar/Baz directory.
D. DEVELOPER can do a GET on resources in the /Bar/Baz directory.
E. DEVELOPER can do only a POST on resources in the /Bar/Baz directory.
F. DEVELOPER can do a TRACE on any resource in the /Bar/Baz directory.

Answer: A, C, D, F

Q7. Which activity supports the data integrity requirements of an application?

A. using HTTPS as a protocol
B. using an LDAP security realm
C. using HTTP Basic authentication

D. using forms-based authentication

Answer: A

Q8. Which mechanism requires the client to provide its public key certificate?

A. HTTP Basic Authentication
B. Form Based Authentication
C. HTTP Digest Authentication
D. HTTPS Client Authentication
Answer: D

Q9. Given the two security constraints in a deployment descriptor:

101. <security-constraint>
102. <!--a correct url-pattern and http-method goes here-->
103. <auth-constraint><role-name>SALES</role-name></auth-
103. <auth-constraint>
104. <role-name>SALES</role-name>
105. </auth-constraint>
106. </security-constraint>
107. <security-constraint>
108. <!--a correct url-pattern and http-method goes here-->
109. <!-- Insert an auth-constraint here -->
110. </security-constraint>

If the two security constraints have the same url-pattern and http-method, which two, inserted independently at line 109, will allow users with role names of either SALES or MARKETING to access
this resource? (Choose two.)

A. <auth-constraint/>
B. <auth-constraint>
<role-name>*</role-name>
</auth-constraint>
C. <auth-constraint>
<role-name>ANY</role-name>
</auth-constraint>
D. <auth-constraint>
<role-name>MARKETING</role-name>
</auth-constraint>

Answer: B, D

Q10. Given this fragment in a servlet:

23. if(req.isUserInRole("Admin")) {
24. // do stuff
25. }
And the following fragment from the related Java EE deployment descriptor:

812. <security-role-ref>
813. <role-name>Admin</role-name>
814. <role-link>Administrator</role-link>
815. </security-role-ref>
900. <security-role>
901. <role-name>Admin</role-name>
902. <role-name>Administrator</role-name>
903. </security-role>
What is the result?
A. Line 24 can never be reached.
B. The deployment descriptor is NOT valid.
C. If line 24 executes, the user's role will be Admin.
D. If line 24 executes, the user's role will be Administrator.
E. If line 24 executes the user's role will NOT be predictable.

Answer: D

Q11.Which two are true about authentication? (Choose two.)
A. Form-based logins should NOT be used with HTTPS.
B. When using Basic Authentication the target server is NOT authenticated.
C. J2EE compliant web containers are NOT required to support the HTTPS protocol.
D. Web containers are required to support unauthenticated access to unprotected web resources.
E. Form-based logins should NOT be used when sessions are maintained by cookies or SSL session information.

Answer: B, D

Q12. If you want to use the Java EE platform's built-in type of authentication that uses a custom HTML page for authentication, which two statements are true? (Choose two.)

A. Your deployment descriptor will need to contain this tag: <auth-method>CUSTOM</auth-method>.
B. The related custom HTML login page must be named loginPage.html.
C. When you use this type of authentication, SSL is turned on automatically.
D. You must have a tag in your deployment descriptor that allows you to point to both a login HTML page and an HTML page for handling any login errors.
E. In the HTML related to authentication for this application, you must use predefined variable names for the variables that store the user and password values.

Answer: D, E

Q13. Given the two security constraints in a deployment descriptor:

101. <security-constraint>
102. <!--a correct url-pattern and http-method goes here-->
103. <auth-constraint><role-name>SALES</role-name></auth-
103. <auth-constraint>
104. <role-name>SALES</role-name>

105. </auth-constraint>
106. </security-constraint>
107. <security-constraint>
108. <!--a correct url-pattern and http-method goes here-->
109. <!-- Insert an auth-constraint here -->
110. </security-constraint>

If the two security constraints have the same url-pattern and http-method, which two, inserted independently at line 109, will allow users with role names of either SALES or MARKETING to access this resource? (Choose two.)

A. <auth-constraint/>
B. <auth-constraint>
<role-name>*</role-name>
</auth-constraint>
C. <auth-constraint>
<role-name>ANY</role-name>
</auth-constraint>
D. <auth-constraint>
<role-name>MARKETING</role-name>
</auth-constraint>

Answer: B, D

Q14. Which two are valid values for the <transport-guarantee> element inside a <security-constraint> element of a web application deployment descriptor? (Choose two.)

A. NULL
B. SECURE
C. INTEGRAL
D. ENCRYPTED
E. CONFIDENTIAL

Answer: C, E

Q15. Which basic authentication type is optional for a J2EE 1.4 compliant web container?

A. HTTP Basic Authentication
B. Form Based Authentication
C. HTTP Digest Authentication
D. HTTPS Client Authentication

Answer: C

Q16. Which security mechanism uses the concept of a realm?

A. authorization
B. data integrity
C. confidentiality
D. authentication

Answer: D

Q17. Which two security mechanisms can be directed through a sub-element of the <user-data-constraint> element in a web application deployment descriptor? (Choose two.)

A. authorization
B. data integrity
C. confidentiality
D. authentication

Answer: B, C

Q18. Which two statements are true about the security-related tags in a valid Java EE deployment descriptor? (Choose two.)

A. Every <security-constraint> tag must have at least one <http-method> tag.
B. A <security-constraint> tag can have many <web-resource-collection> tags.
C. A given <auth-constraint> tag can apply to only one <web-resource-collection> tag.
D. A given <web-resource-collection> tag can contain from zero to many <url-pattern> tags.
E. It is possible to construct a valid <security-constraint> tag such that, for a given resource, no user roles can access that resource.

Answer: B, E

Q19. Which element of a web application deployment descriptor <security-constraint> element is required?

A. <realm-name>
B. <auth-method>
C. <security-role>
D. <transport-guarantee>
E. <web-resource-collection>

Answer: E

Q 20 Which two are required elements for the <web-resource-collection> element of a web application deployment descriptor? (Choose two.)
A. <realm-name>
B. <url-pattern>
C. <description>
D. <web-resource-name>
E. <transport-guarantee>
Answer: B, D

Q21. Given:

```
3. class MyServlet extends HttpServlet {
4. public void doPut(HttpServletRequest req,
HttpServletResponse resp)
throws ServletException, IOException {
5. // servlet code here
...
26. }
27. }
```

If the DD contains a single security constraint associated with MyServlet and its only <http-method> tags and <auth-constraint> tags are:

```
<http-method>GET</http-method>
<http-method>PUT</http-method>
<auth-constraint>Admin</auth-constraint>
```

Which four requests would be allowed by the container? (Choose four.)

A. A user whose role is Admin can perform a PUT.
B. A user whose role is Admin can perform a GET.
C. A user whose role is Admin can perform a POST.
D. A user whose role is Member can perform a PUT.
E. A user whose role is Member can perform a POST.
F. A user whose role is Member can perform a GET.

Answer: A, B, C, E

Q22. What is true about Java EE authentication mechanisms?

A. If your deployment descriptor correctly declares an authentication type of CLIENT_CERT, your users must have a certificate from an official source before they can use your application.
B. If your deployment descriptor correctly declares an authentication type of BASIC, the container automatically requests a user name and password whenever a user starts a new session.
C. If you want your web application to support the widest possible array of browsers, and you want to perform authentication, the best choice of Java EE authentication mechanisms is DIGEST.
D. To use Java EE FORM authentication, you must declare two HTML files in your deployment descriptor, and you must use a predefined action in the HTML file that handles your user's login.
Answer: D

Q23. Which two statements are true about using the isUserInRole method to implement security in a Java EE application? (Choose two.)

A. It can be invoked only from the doGet or doPost methods.
B. It can be used independently of the getRemoteUser method.
C. Can return "true" even when its argument is NOT defined as a valid role name in the deployment descriptor.
D. Using the isUserInRole method overrides any declarative authentication related to the method in which it is invoked.
E. Using the isUserInRole method overrides any declarative authorization related to the method in which it is invoked.

Answer: B, C

Q24.developer has used this code within a servlet:
62. if(request.isUserInRole("vip")) {
63. // VIP-related logic here
64. }
What else must the developer do to ensure that the intended security goal is achieved?
A. create a user called vip in the security realm
B. define a group within the security realm and call it vip
C. define a security-role named vip in the deployment descriptor
D. declare a security-role-ref for vip in the deployment descriptor
Answer: D

Unit-5 : Web Application Security

Objectives

1. Based on the servlet specification, compare and contrast the following security mechanisms: (a) authentication, (b) authorization, (c) data integrity, and (d) confidentiality.
2. In the deployment descriptor, declare a security constraint, a Web resource, the transport guarantee, the login configuration, and a security role.
3. Compare and contrast the authentication types (BASIC, DIGEST, FORM, and CLIENT-CERT); describe how the type works; and given a scenario, select an appropriate type.

Q1. Given:

```
3. class MyServlet extends HttpServlet {
4. public void doPut(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
5. // servlet code here ...
26. }
27. }
```

If the DD contains a single security constraint associated with MyServlet and its only <http-method> tags
and <auth-constraint> tags are:
<http-method>GET</http-method>
<http-method>PUT</http-method>
<auth-constraint>Admin</auth-constraint>

Which four requests would be allowed by the container? (Choose four.)

A. A user whose role is Admin can perform a PUT.
B. A user whose role is Admin can perform a GET.
C. A user whose role is Admin can perform a POST.
D. A user whose role is Member can perform a PUT.
E. A user whose role is Member can perform a POST.
F. A user whose role is Member can perform a GET.

Answer: A, B, C, E
Q2. What is true about Java EE authentication mechanisms?
A. If your deployment descriptor correctly declares an authentication type of CLIENT_CERT, your users must have a certificate from an official source before they can use your application.

B. If your deployment descriptor correctly declares an authentication type of BASIC, the container automatically requests a user name and password whenever a user starts a new session.

C. If you want your web application to support the widest possible array of browsers, and you want to perform authentication, the best choice of Java EE authentication mechanisms is DIGEST.

D. To use Java EE FORM authentication, you must declare two HTML files in your deployment descriptor, and you must use a predefined action in the HTML file that handles your user's login.
Answer: D

Q3. If you want to use the Java EE platform's built-in type of authentication that uses a custom HTML page for authentication, which two statements are true? (Choose two.)

A. Your deployment descriptor will need to contain this tag:
<auth-method>CUSTOM</auth-method>.
B. The related custom HTML login page must be named loginPage.html.
C. When you use this type of authentication, SSL is turned on automatically.
D. You must have a tag in your deployment descriptor that allows you to point to both a login HTML page and an HTML page for handling any login errors.
E. In the HTML related to authentication for this application, you must use predefined variable names for the variables that store the user and password values.
Answer: D, E

Q4. Given this fragment in a servlet:
23. if(req.isUserInRole("Admin")) {
24. // do stuff
25. }
And the following fragment from the related Java EE deployment descriptor:
812. <security-role-ref>
813. <role-name>Admin</role-name>
814. <role-link>Administrator</role-link>
815. </security-role-ref>
900. <security-role>
901. <role-name>Admin</role-name>
902. <role-name>Administrator</role-name>
903. </security-role>
What is the result?

A. Line 24 can never be reached.
B. The deployment descriptor is NOT valid.
C. If line 24 executes, the user's role will be Admin.
D. If line 24 executes, the user's role will be Administrator.
E. If line 24 executes the user's role will NOT be predictable.
Answer: D

Q5. Given the security constraint in a DD:

101. <security-constraint>
102. <web-resource-collection>
103. <web-resource-name>Foo</web-resource-name>
104. <url-pattern>/Bar/Baz/*</url-pattern>
105. <http-method>POST</http-method>
106. </web-resource-collection>
107. <auth-constraint>
108. <role-name>DEVELOPER</role-name>

109. </auth-constraint>
110. </security-constraint>

And given that "MANAGER" is a valid role-name, which four are true for this security constraint?(Choose four.)

A. MANAGER can do a GET on resources in the /Bar/Baz directory.
B. MANAGER can do a POST on any resource in the /Bar/Baz directory.
C. MANAGER can do a TRACE on any resource in the /Bar/Baz directory.
D. DEVELOPER can do a GET on resources in the /Bar/Baz directory.
E. DEVELOPER can do only a POST on resources in the /Bar/Baz directory.
F. DEVELOPER can do a TRACE on any resource in the /Bar/Baz directory.

Answer: A, C, D, F

Q6. Given the security constraint in a DD:

101. <security-constraint>
102. <web-resource-collection>
103. <web-resource-name>Foo</web-resource-name>
104. <url-pattern>/Bar/Baz/*</url-pattern>
105. <http-method>POST</http-method>
106. </web-resource-collection>
107. <auth-constraint>
108. <role-name>DEVELOPER</role-name>
109. </auth-constraint>
110. </security-constraint>
And given that "MANAGER" is a valid role-name, which four are true for this security constraint?(Choose four.)

A. MANAGER can do a GET on resources in the /Bar/Baz directory.
B. MANAGER can do a POST on any resource in the /Bar/Baz directory.
C. MANAGER can do a TRACE on any resource in the /Bar/Baz directory.
D. DEVELOPER can do a GET on resources in the /Bar/Baz directory.
E. DEVELOPER can do only a POST on resources in the /Bar/Baz directory.
F. DEVELOPER can do a TRACE on any resource in the /Bar/Baz directory.

Answer: A, C, D, F

Q7. Which activity supports the data integrity requirements of an application?

A. using HTTPS as a protocol
B. using an LDAP security realm
C. using HTTP Basic authentication
D. using forms-based authentication

Answer: A

Q8. Which mechanism requires the client to provide its public key certificate?

A. HTTP Basic Authentication
B. Form Based Authentication
C. HTTP Digest Authentication
D. HTTPS Client Authentication
Answer: D

Q9. Given the two security constraints in a deployment descriptor:

101. <security-constraint>
102. <!--a correct url-pattern and http-method goes here-->
103. <auth-constraint><role-name>SALES</role-name></auth-
103. <auth-constraint>
104. <role-name>SALES</role-name>
105. </auth-constraint>
106. </security-constraint>
107. <security-constraint>
108. <!--a correct url-pattern and http-method goes here-->
109. <!-- Insert an auth-constraint here -->
110. </security-constraint>

If the two security constraints have the same url-pattern and http-method, which two, inserted independently at line 109, will allow users with role names of either SALES or MARKETING to access
this resource? (Choose two.)

A. <auth-constraint/>
B. <auth-constraint>
<role-name>*</role-name>
</auth-constraint>
C. <auth-constraint>
<role-name>ANY</role-name>
</auth-constraint>
D. <auth-constraint>
<role-name>MARKETING</role-name>
</auth-constraint>

Answer: B, D

Q10. Given this fragment in a servlet:

23. if(req.isUserInRole("Admin")) {
24. // do stuff
25. }
And the following fragment from the related Java EE deployment descriptor:

812. <security-role-ref>
813. <role-name>Admin</role-name>
814. <role-link>Administrator</role-link>
815. </security-role-ref>
900. <security-role>

901. <role-name>Admin</role-name>
902. <role-name>Administrator</role-name>
903. </security-role>
What is the result?
A. Line 24 can never be reached.
B. The deployment descriptor is NOT valid.
C. If line 24 executes, the user's role will be Admin.
D. If line 24 executes, the user's role will be Administrator.
E. If line 24 executes the user's role will NOT be predictable.

Answer: D

Q11.Which two are true about authentication? (Choose two.)
A. Form-based logins should NOT be used with HTTPS.
B. When using Basic Authentication the target server is NOT authenticated.
C. J2EE compliant web containers are NOT required to support the HTTPS protocol.
D. Web containers are required to support unauthenticated access to unprotected web resources.
E. Form-based logins should NOT be used when sessions are maintained by cookies or SSL session information.

Answer: B, D

Q12. If you want to use the Java EE platform's built-in type of authentication that uses a custom HTML page for authentication, which two statements are true? (Choose two.)

A. Your deployment descriptor will need to contain this tag: <auth-method>CUSTOM</auth-method>.
B. The related custom HTML login page must be named loginPage.html.
C. When you use this type of authentication, SSL is turned on automatically.
D. You must have a tag in your deployment descriptor that allows you to point to both a login HTML page and an HTML page for handling any login errors.
E. In the HTML related to authentication for this application, you must use predefined variable names for the variables that store the user and password values.

Answer: D, E

Q13. Given the two security constraints in a deployment descriptor:

101. <security-constraint>
102. <!--a correct url-pattern and http-method goes here-->
103. <auth-constraint><role-name>SALES</role-name></auth-
103. <auth-constraint>
104. <role-name>SALES</role-name>
105. </auth-constraint>
106. </security-constraint>
107. <security-constraint>
108. <!--a correct url-pattern and http-method goes here-->
109. <!-- Insert an auth-constraint here -->

110. </security-constraint>

If the two security constraints have the same url-pattern and http-method, which two, inserted independently at line 109, will allow users with role names of either SALES or MARKETING to access this resource? (Choose two.)

A. <auth-constraint/>
B. <auth-constraint>
<role-name>*</role-name>
</auth-constraint>
C. <auth-constraint>
<role-name>ANY</role-name>
</auth-constraint>
D. <auth-constraint>
<role-name>MARKETING</role-name>
</auth-constraint>

Answer: B, D

Q14. Which two are valid values for the <transport-guarantee> element inside a <security-constraint> element of a web application deployment descriptor? (Choose two.)

A. NULL
B. SECURE
C. INTEGRAL
D. ENCRYPTED
E. CONFIDENTIAL

Answer: C, E

Q15. Which basic authentication type is optional for a J2EE 1.4 compliant web container?

A. HTTP Basic Authentication
B. Form Based Authentication
C. HTTP Digest Authentication
D. HTTPS Client Authentication

Answer: C

Q16. Which security mechanism uses the concept of a realm?

A. authorization
B. data integrity
C. confidentiality
D. authentication

Answer: D

Q17. Which two security mechanisms can be directed through a sub-element of the <user-data-constraint> element in a web application deployment descriptor? (Choose two.)

A. authorization
B. data integrity
C. confidentiality
D. authentication

Answer: B, C

Q18. Which two statements are true about the security-related tags in a valid Java EE deployment descriptor? (Choose two.)

A. Every <security-constraint> tag must have at least one <http-method> tag.
B. A <security-constraint> tag can have many <web-resource-collection> tags.
C. A given <auth-constraint> tag can apply to only one <web-resource-collection> tag.
D. A given <web-resource-collection> tag can contain from zero to many <url-pattern> tags.
E. It is possible to construct a valid <security-constraint> tag such that, for a given resource, no user roles can access that resource.

Answer: B, E

Q19. Which element of a web application deployment descriptor <security-constraint> element is required?

A. <realm-name> **B. <auth-method>**
C. <security-role> **D. <transport-guarantee>**
E. <web-resource-collection>

Answer: E

Q 20 Which two are required elements for the <web-resource-collection> element of a web application deployment descriptor? (Choose two.)
A. <realm-name> **B. <url-pattern>**
C. <description> **D. <web-resource-name>**
E. <transport-guarantee>
Answer: B, D

Q21. Given:

```
3. class MyServlet extends HttpServlet {
4. public void doPut(HttpServletRequest req,
HttpServletResponse resp)
throws ServletException, IOException {
5. // servlet code here
...
26. }
27. }
```

If the DD contains a single security constraint associated with MyServlet and its only <http-method> tags and <auth-constraint> tags are:
<http-method>GET</http-method>
<http-method>PUT</http-method>
<auth-constraint>Admin</auth-constraint>

Which four requests would be allowed by the container? (Choose four.)

A. A user whose role is Admin can perform a PUT.
B. A user whose role is Admin can perform a GET.

C. A user whose role is Admin can perform a POST.
D. A user whose role is Member can perform a PUT.
E. A user whose role is Member can perform a POST.
F. A user whose role is Member can perform a GET.

Answer: A, B, C, E

Q22. What is true about Java EE authentication mechanisms?

A. If your deployment descriptor correctly declares an authentication type of CLIENT_CERT, your users must have a certificate from an official source before they can use your application.
B. If your deployment descriptor correctly declares an authentication type of BASIC, the container automatically requests a user name and password whenever a user starts a new session.
C. If you want your web application to support the widest possible array of browsers, and you want to perform authentication, the best choice of Java EE authentication mechanisms is DIGEST.
D. To use Java EE FORM authentication, you must declare two HTML files in your deployment descriptor, and you must use a predefined action in the HTML file that handles your user's login.
Answer: D

Q23. Which two statements are true about using the isUserInRole method to implement security in a Java EE application? (Choose two.)

A. It can be invoked only from the doGet or doPost methods.
B. It can be used independently of the getRemoteUser method.
C. Can return "true" even when its argument is NOT defined as a valid role name in the deployment descriptor.
D. Using the isUserInRole method overrides any declarative authentication related to the method in which it is invoked.
E. Using the isUserInRole method overrides any declarative authorization related to the method in which it is invoked.

Answer: B, C

Q24.developer has used this code within a servlet:
62. if(request.isUserInRole("vip")) {
63. // VIP-related logic here
64. }
What else must the developer do to ensure that the intended security goal is achieved?

A. create a user called vip in the security realm
B. define a group within the security realm and call it vip
C. define a security-role named vip in the deployment descriptor
D. declare a security-role-ref for vip in the deployment descriptor

Answer: D