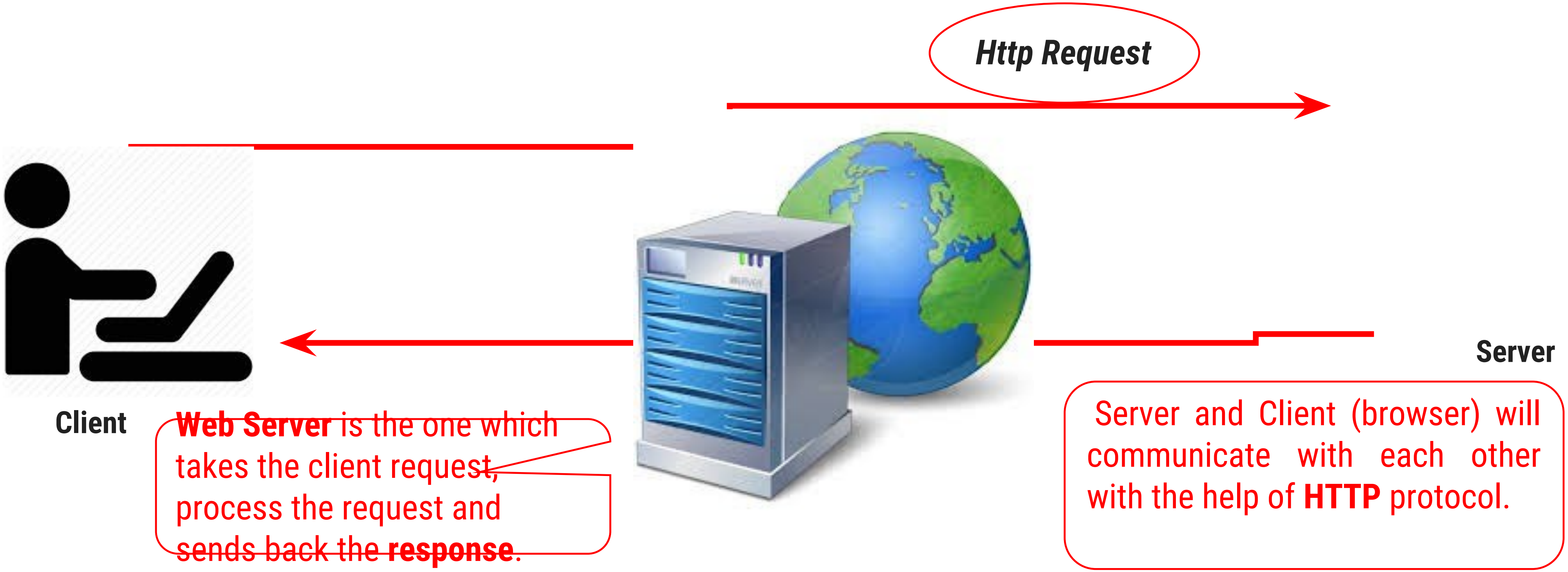
What is Servlet?

- Servlet is **java class** which extends the functionality of **web server** by dynamically generating **web pages**.

- **Servlet: Basic Terms**

  - Before looking into Servlet, we will see some important keywords about web application.



Web Client: We will call **browsers** (IE, Chrome, Mozilla etc.) as a **Client**, which helps in communicating with the server

Web Server is the one which takes the client request, process the request and sends back the **response**.

Server and Client (browser) will communicate with each other with the help of **HTTP** protocol.

Introduction

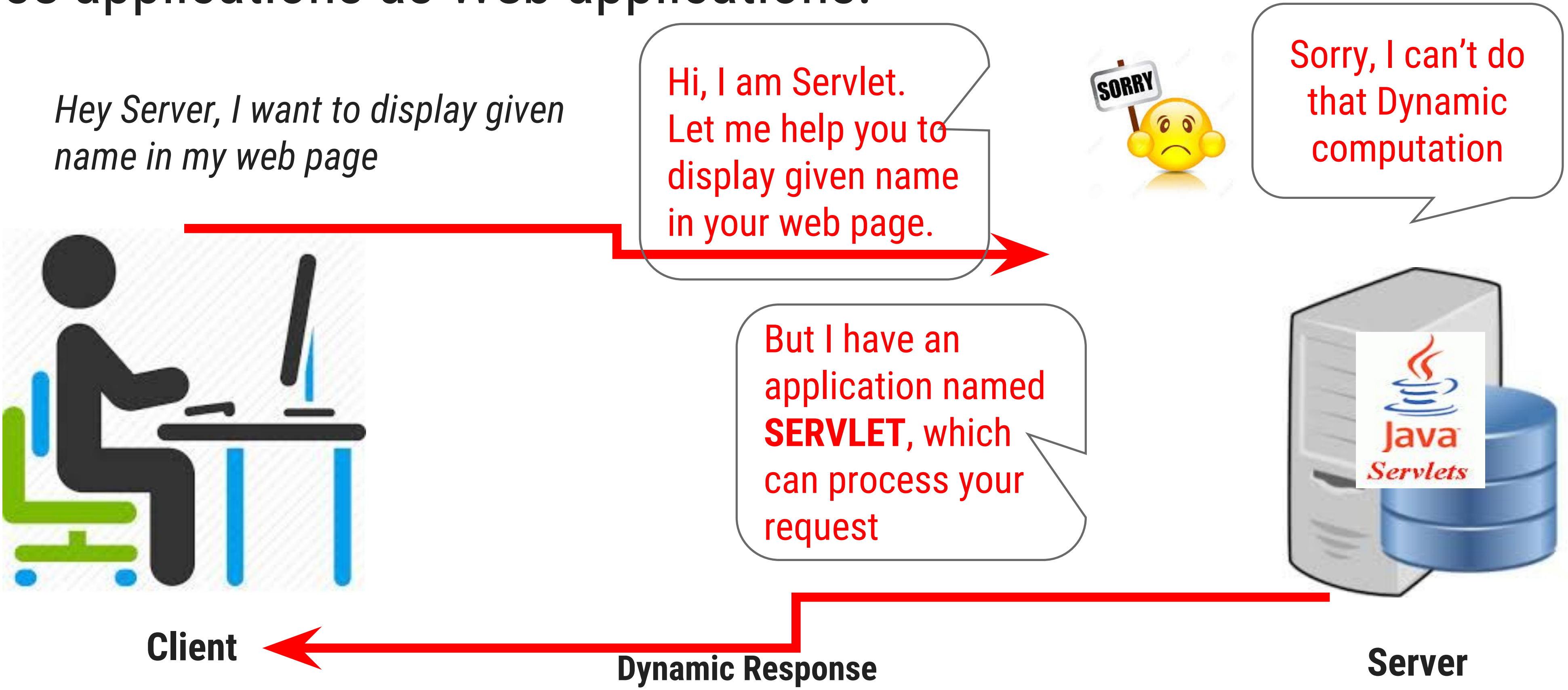
- Servlet technology is used to create **Dynamic** web application
- Servlet technology is robust and scalable .
- Before Servlet, CGI (Common Gateway Interface) scripting language was popular as a server-side programming language, but there were many disadvantages of this technology.

Changes with respect to time

1. To retrive server's current DATE and Time
2. News paper clippings
3. Online Shopping
   e.g. Virtual Dressing Room
   ..
   .

Why we need Servlet?

- Nowadays everything is available on Internet.
- Starting from e-banking, e-commerce everything is available through internet. We call all these applications as Web applications.



Hey Server, I want to display given name in my web page

Hi, I am Servlet. Let me help you to display given name in your web page.

Sorry, I can’t do that Dynamic computation

But I have an application named **SERVLET**, which can process your request

Client

Dynamic Response

Server

Scripting Language

Server-Side Scripting Language

PHP
ASP.NET
(C# OR Visual Basic)
C++
Java and JSP
Python
Ruby on Rails etc.

Server-side scripting is often used to provide a customized interface for the user.

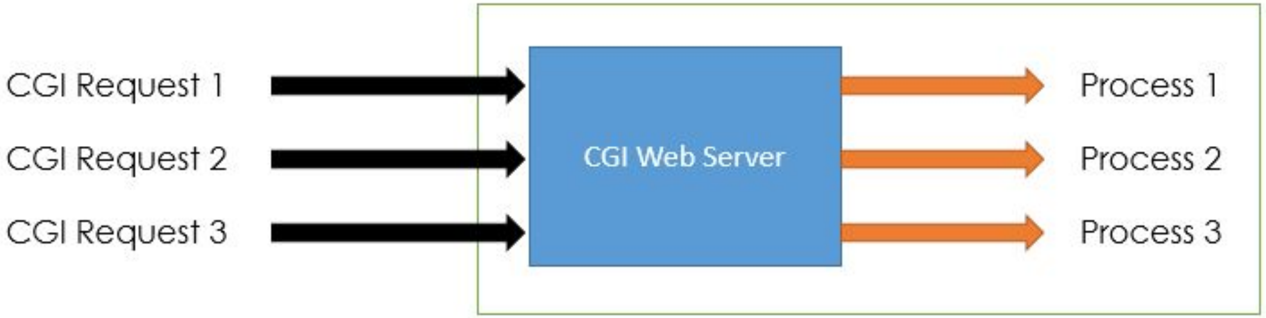
Client-Side Scripting Language

JavaScript
VBScript
HTML (Structure)
CSS (Designing)
AJAX
jQuery etc.

Client-side scripting is an important part of the Dynamic HTML. Usually run on client's browser.

CGI (Common Gateway Interface)

- CGI was the 1st server-side scripting technique for creating dynamic content.
- CGI is used to execute a program that resides in the server to process data or access databases to produce the relevant dynamic content.
- For each request CGI Server receives, It creates new Operating System Process.



- If the number of requests from the client increases then more time it will take to respond to the request.
- As programs executed by CGI Script are written in the native languages such as C, C++, perl which are not portable.

Comparing Servlet with CGI

- CGI programs are used to execute programs written inside the native language.
- While in Servlet, all the programs are compiled into the Java bytecode, which is then run in the Java virtual machine.
- In Servlet, all the requests coming from the Client are processed with the threads instead of the OS process.

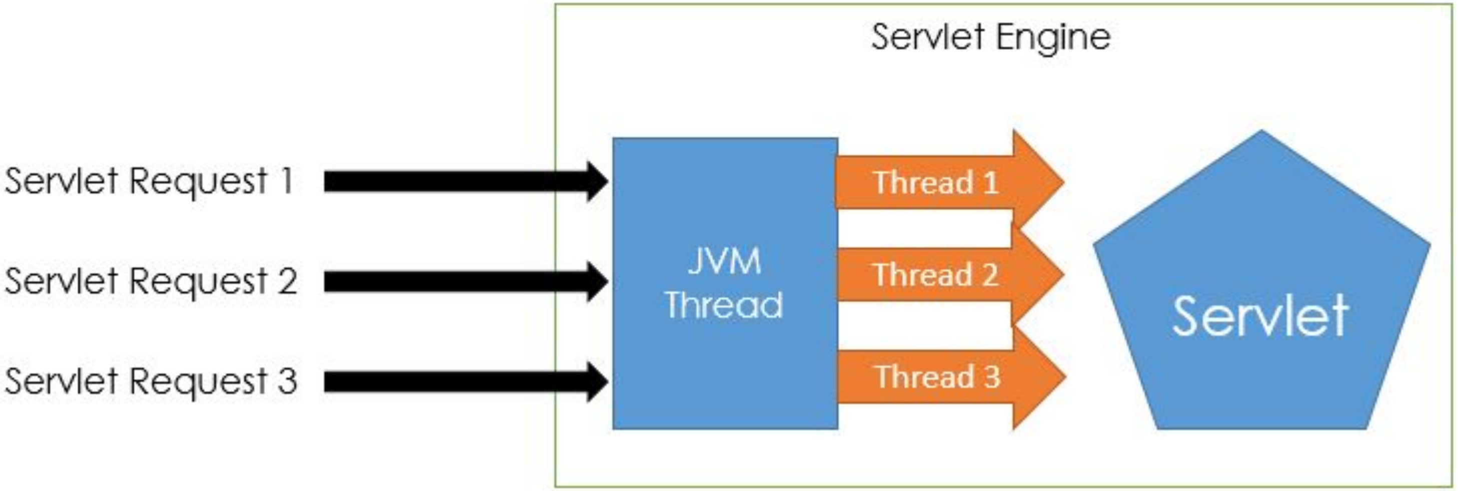
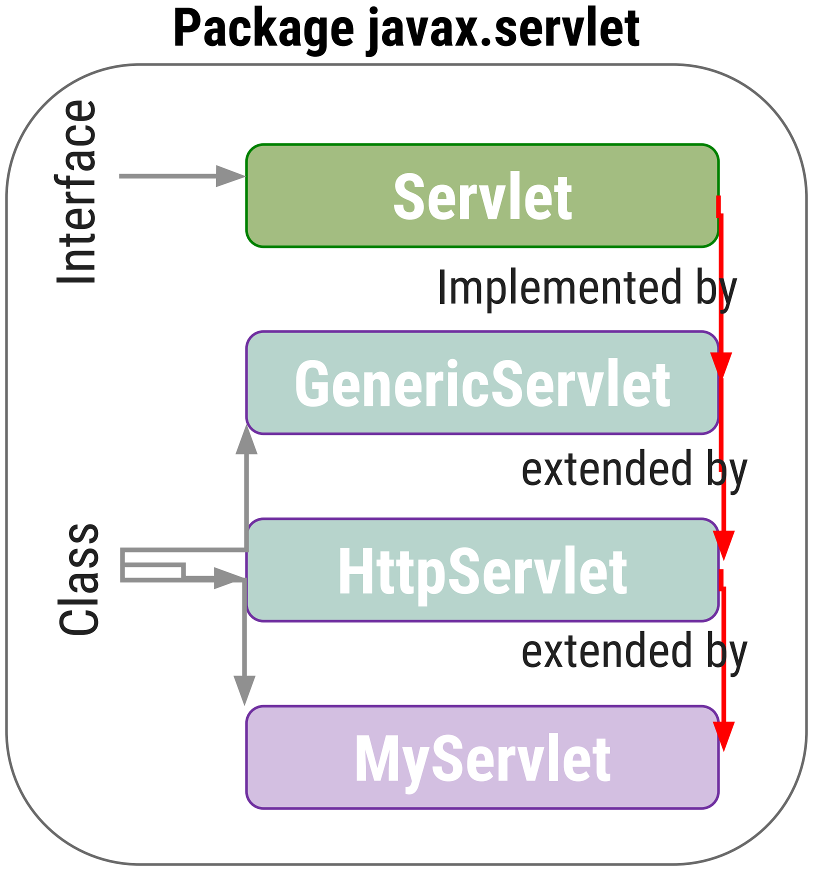


Image Reference:http://www.c4learn.com/java/servlet/servlet-vs-cgi/

Summary: CGI vs Servlet

CGI	Servlet
CGI was not portable.	Servlets are portable.
In CGI each request is handled by heavy weight OS process.	In Servlets each request is handled by lightweight Java Thread.
Session tracking and caching of previous computations cannot be performed.	Session tracking and caching of previous computations can be performed
CGI cannot handle cookies.	Servlets can handle cookies.
CGI does not provide sharing property.	Servlets can share data among each other.
CGI is more expensive than Servlets	Servlets is inexpensive than CGI.

Servlet Packages



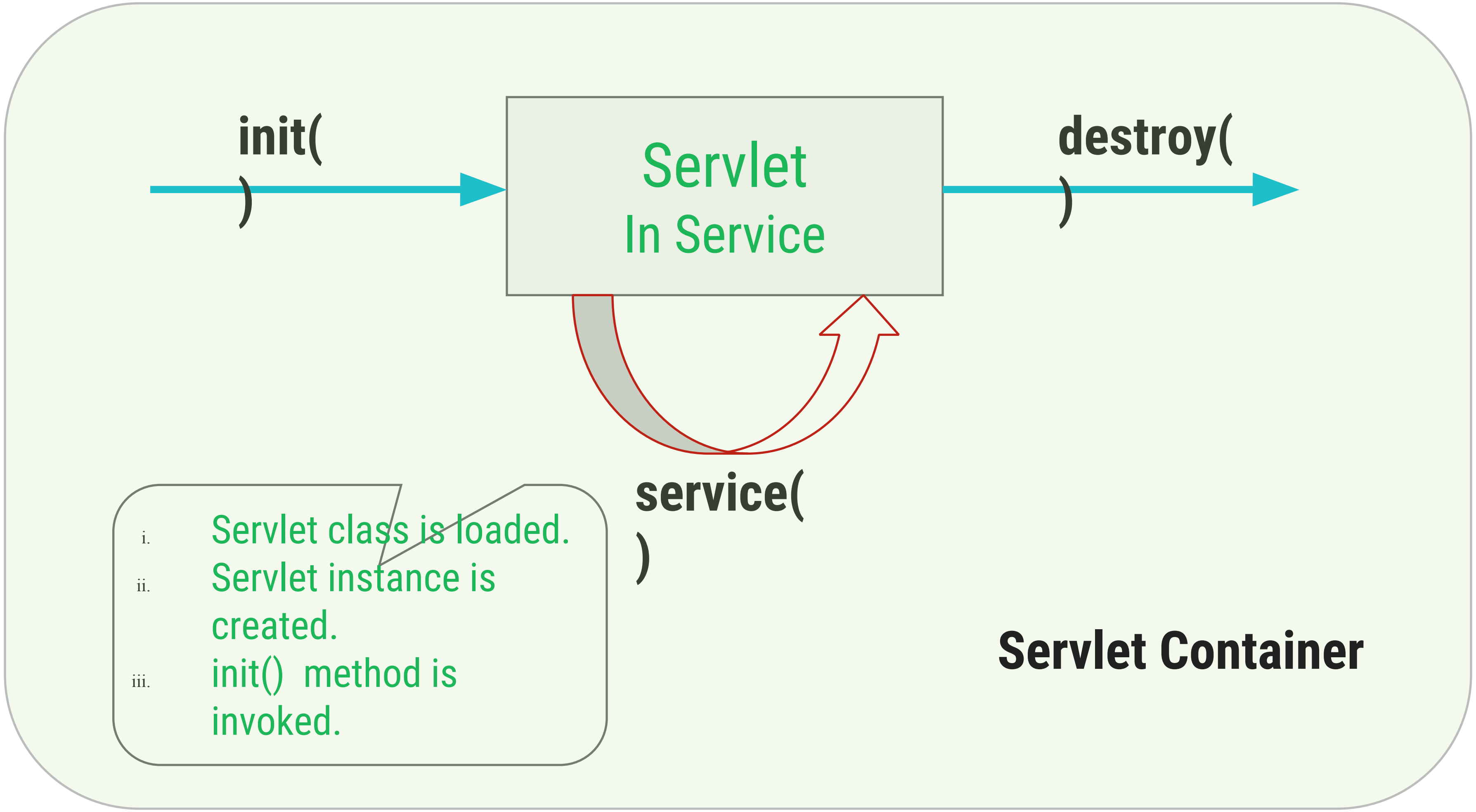
Servlet interface needs to be implemented for creating any servlet. It provides 3 life cycle methods.

It provides implementation of methods of Servlet interfaces.

Contains interface and abstract class for servlet that understands HTTP protocol.
Package: javax.servlet.http

User defined Servlet class.

Servlet Life Cycle



init(
)

Servlet

In Service

destroy(
)

service(
)

i. Servlet class is loaded.

ii. Servlet instance is created.

iii. init() method is invoked.

Servlet Container

Servlet Life Cycle: init()

- Servlet class is loaded
  - The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.
- Servlet instance is created
  - The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.
- Init() method is invoked
  - The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet.

Syntax

```
public void init(ServletConfig config) throws ServletException
{
//initialization…
}
```

A servlet configuration object used by a servlet container to pass information to a servlet during initialization process.

Servlet Life Cycle: Service()

- The service() method is the main method to perform the actual task.
- The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the response back to the client.
- Each time the server receives a request for a servlet, the server spawns a new thread and calls service.

Syntax

```
public void service(ServletRequest request, ServletResponse response)
                              throws ServletException, IOException
{
    …
}
```

- The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.
- The doGet() and doPost() are most frequently used methods with in each service request.

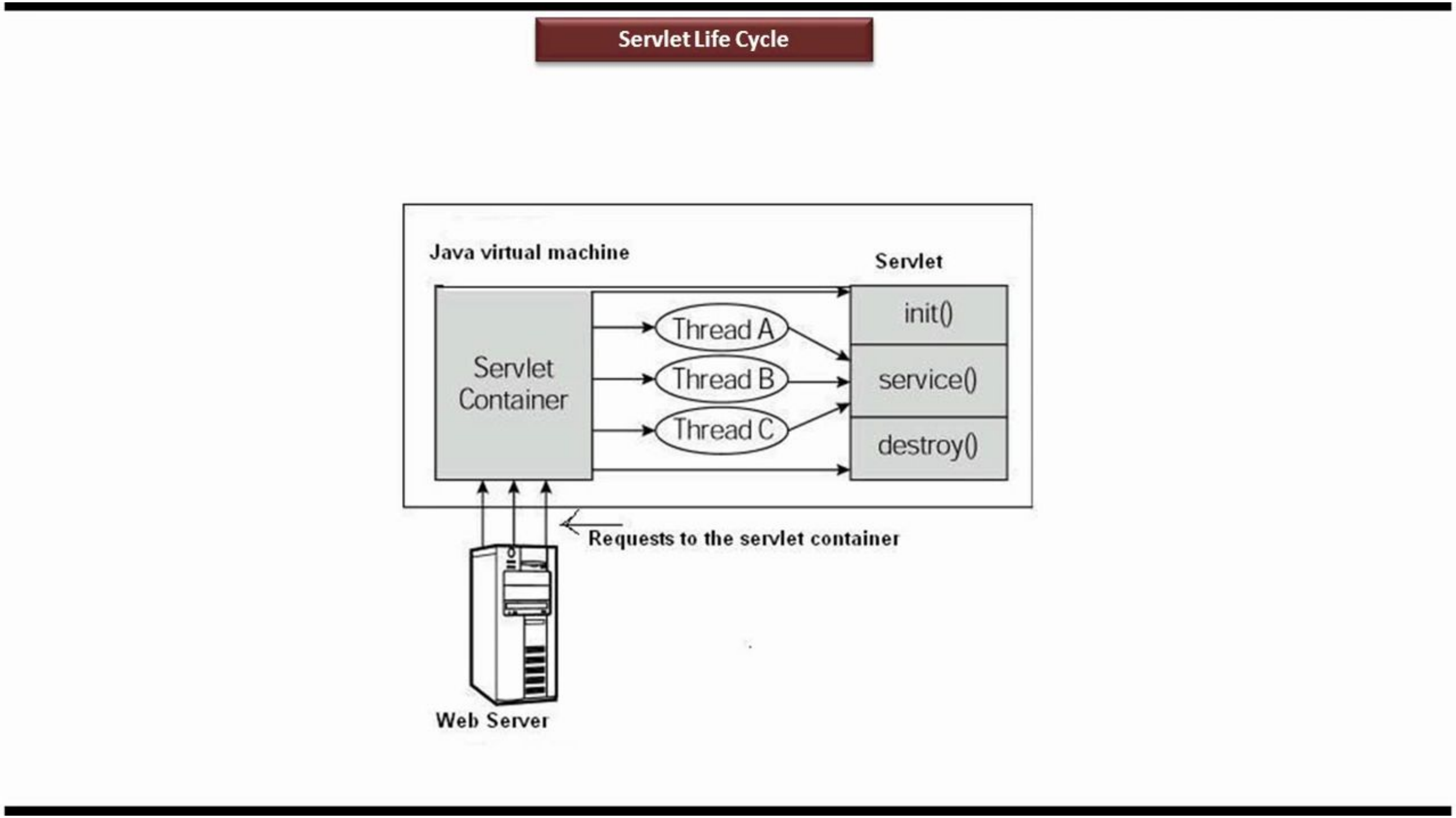
Servlet Life Cycle: Destroy()

- The destroy() method is called only once at the end of the life cycle of a servlet.
- This method gives your servlet a chance to close
  - database connections,
  - halt background threads,
  - write cookie lists or hit counts to disk, and
  - perform other such cleanup activities.
- After the destroy() method is called, the servlet object is marked for garbage collection.

Syntax

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

Servlet Life Cycle



Servlet Life Cycle

doGet() v/s doPost()

- doGet()
  - A GET request results from request for a URL or from an HTML form, should be handled by doGet() method.

Syntax

```
public void doGet(HttpServletRequest request,HttpServletResponse response)
                             throws ServletException, IOException
{
    // Servlet code …
}
```

Syntax

```
public void doGet(HttpServletRequest request,HttpServletResponse response)
                throws ServletException, IOException
{
    // Servlet code …
}
```

- doPost()
  - A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

doGet() vs doPost()

doGet()	doPost()
In this method, parameters are appended to the URL and sent along with header information	In doPost(), parameters are sent in separate line in the body
Maximum size of data that can be sent using doGet() is 240 bytes	There is no maximum size for data
Parameters are not encrypted	Parameters are encrypted here
Application: Used when small amount of insensitive data like a query has to be sent as a request. *It is default method.*	***Application***: Used when comparatively large amount of sensitive data has to be sent. E.g. submitting sign_in or login form.
doGet() is faster comparatively	doPost() is slower compared to doGet() since doPost() does not write the content length

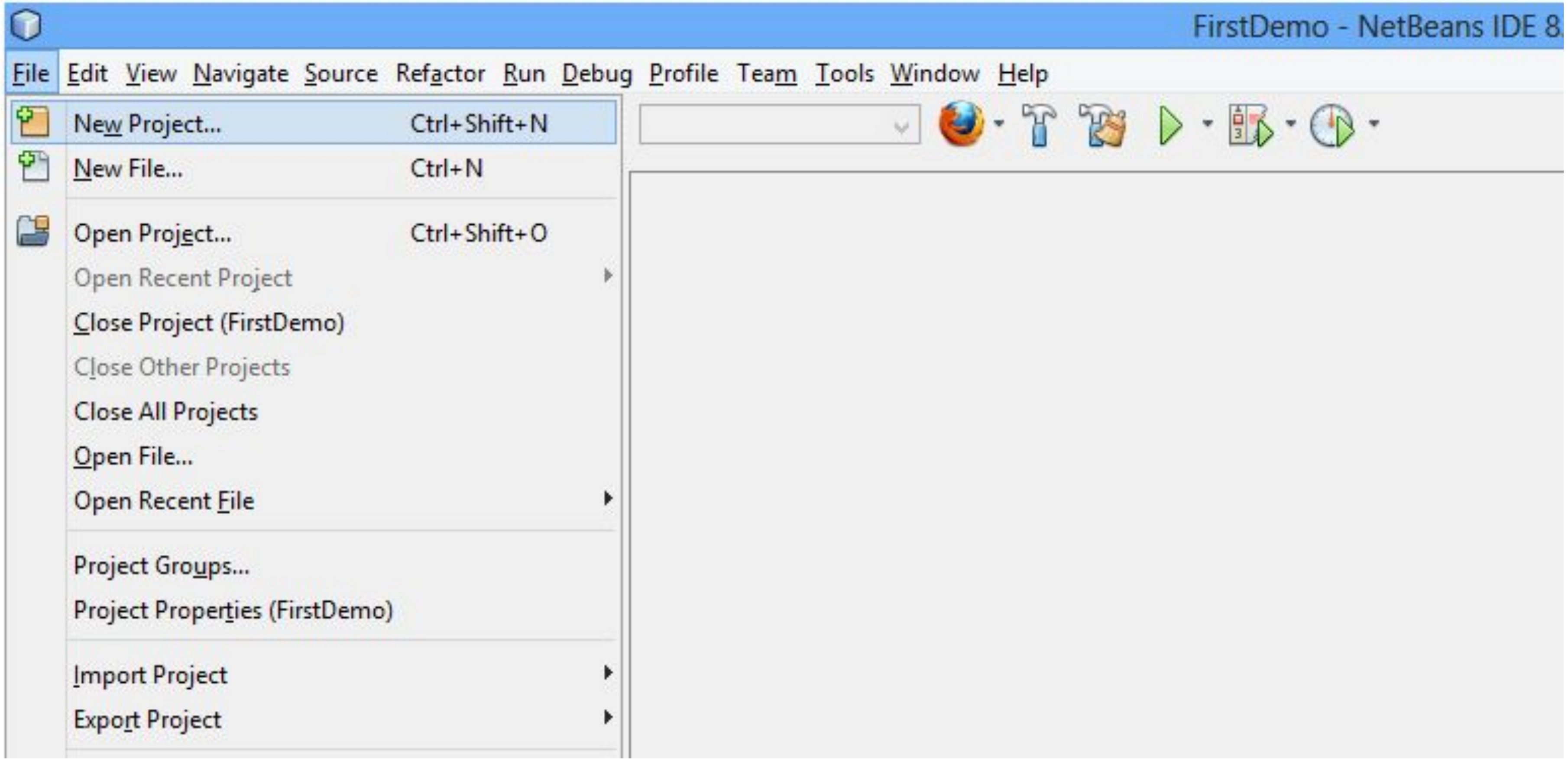
Servlet Life Cycle: Servlet Code

MyServlet.Java

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

public class MyServlet1 extends GenericServlet
{
    public void init() throws ServletException
    {//Initailization Code
    }

    public void service(ServletRequest request,ServletResponse response)   throws
ServletException,IOException
    {//Servlet code
    }

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

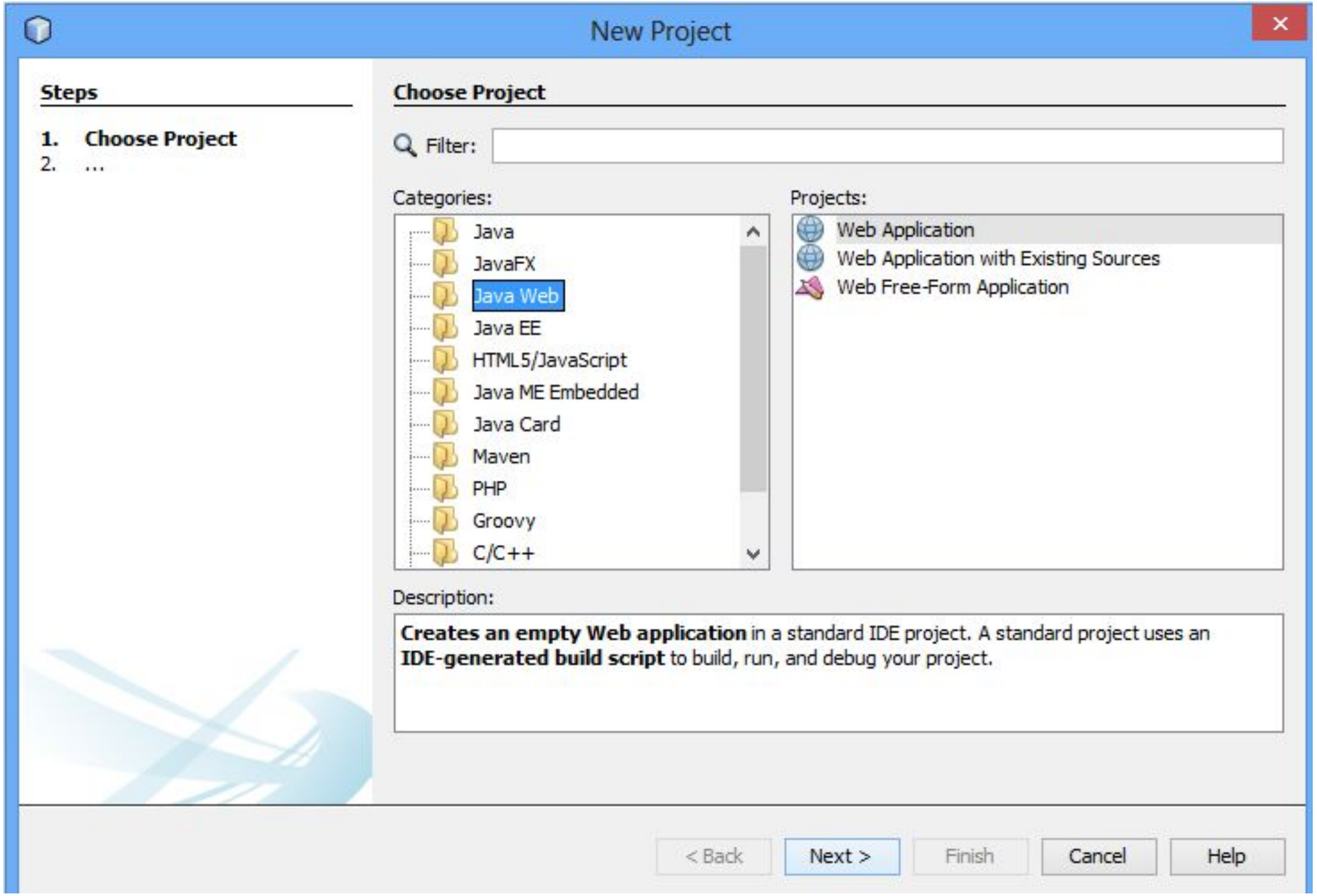
Steps to run Servlet Program in Netbeans

- Step 1: Open Netbeans IDE, Select **File -> New Project**


FirstDemo - NetBeans IDE 8
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
New Project... Ctrl+Shift+N
New File... Ctrl+N
Open Project... Ctrl+Shift+O
Open Recent Project
Close Project (FirstDemo)
Close Other Projects
Close All Projects
Open File...
Open Recent File
Project Groups...
Project Properties (FirstDemo)
Import Project
Export Project

Steps for Servlet Program

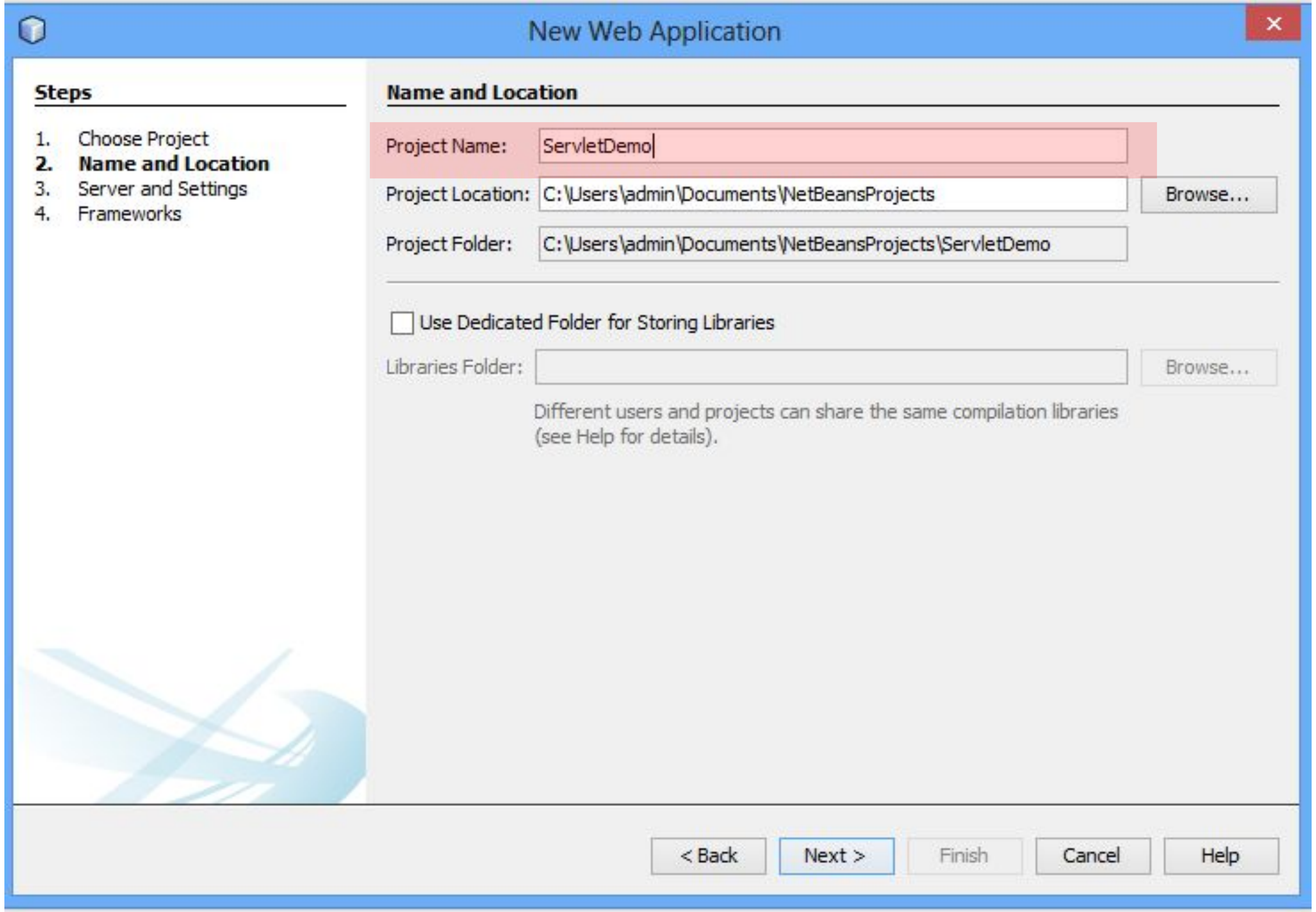
Step 2: Select **Java Web** -> **Web Application**, then click on Next



Creates an empty Web application in a standard IDE project. A standard project uses an **IDE-generated build script** to build, run, and debug your project.

Steps for Servlet Program

- Step 3: Give a name to your project and click on Next,



New Web Application

Steps

1. Choose Project
2. **Name and Location**
3. Server and Settings
4. Frameworks

Name and Location

Project Name: ServletDemo

Project Location: C:\Users\admin\Documents\NetBeansProjects

Project Folder: C:\Users\admin\Documents\NetBeansProjects\ServletDemo

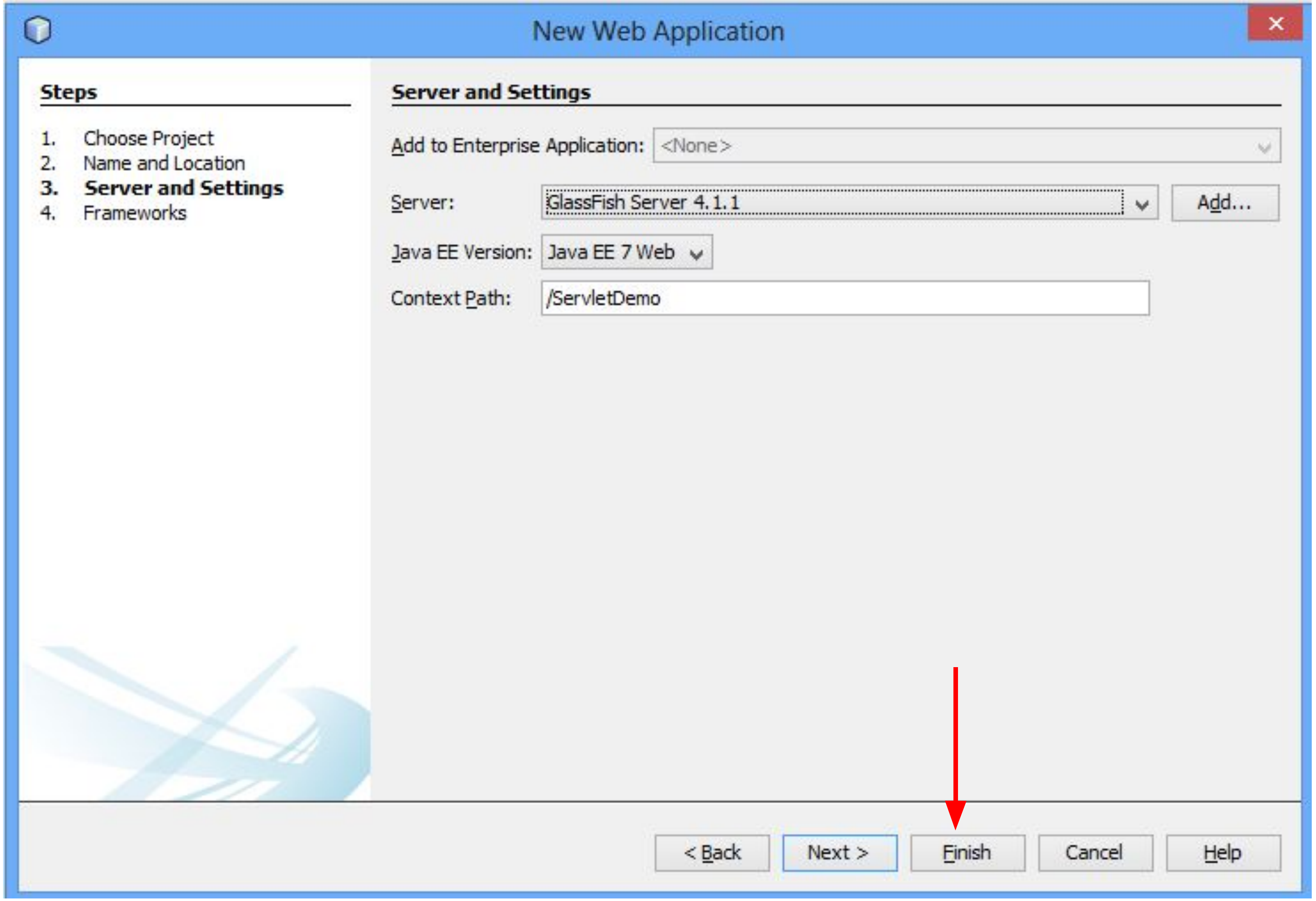
□ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

Steps for Servlet Program

- Step 4: and then, Click **Finish**



New Web Application

Steps

1. Choose Project
2. Name and Location
3. **Server and Settings**
4. Frameworks

Server and Settings

Add to Enterprise Application: <None>

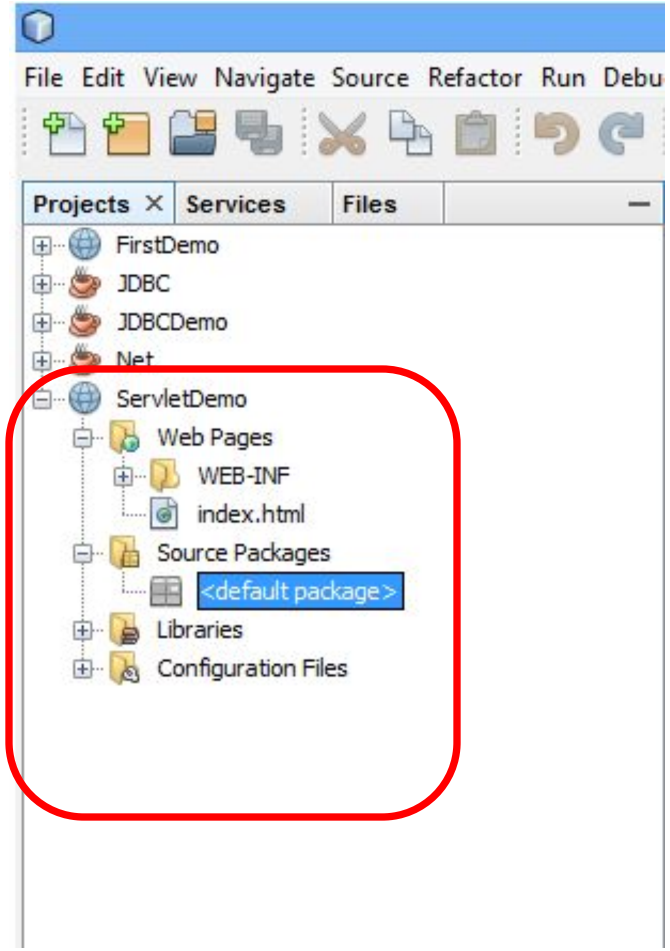
Server: GlassFish Server 4.1.1

Java EE Version: Java EE 7 Web

Context Path: /ServletDemo

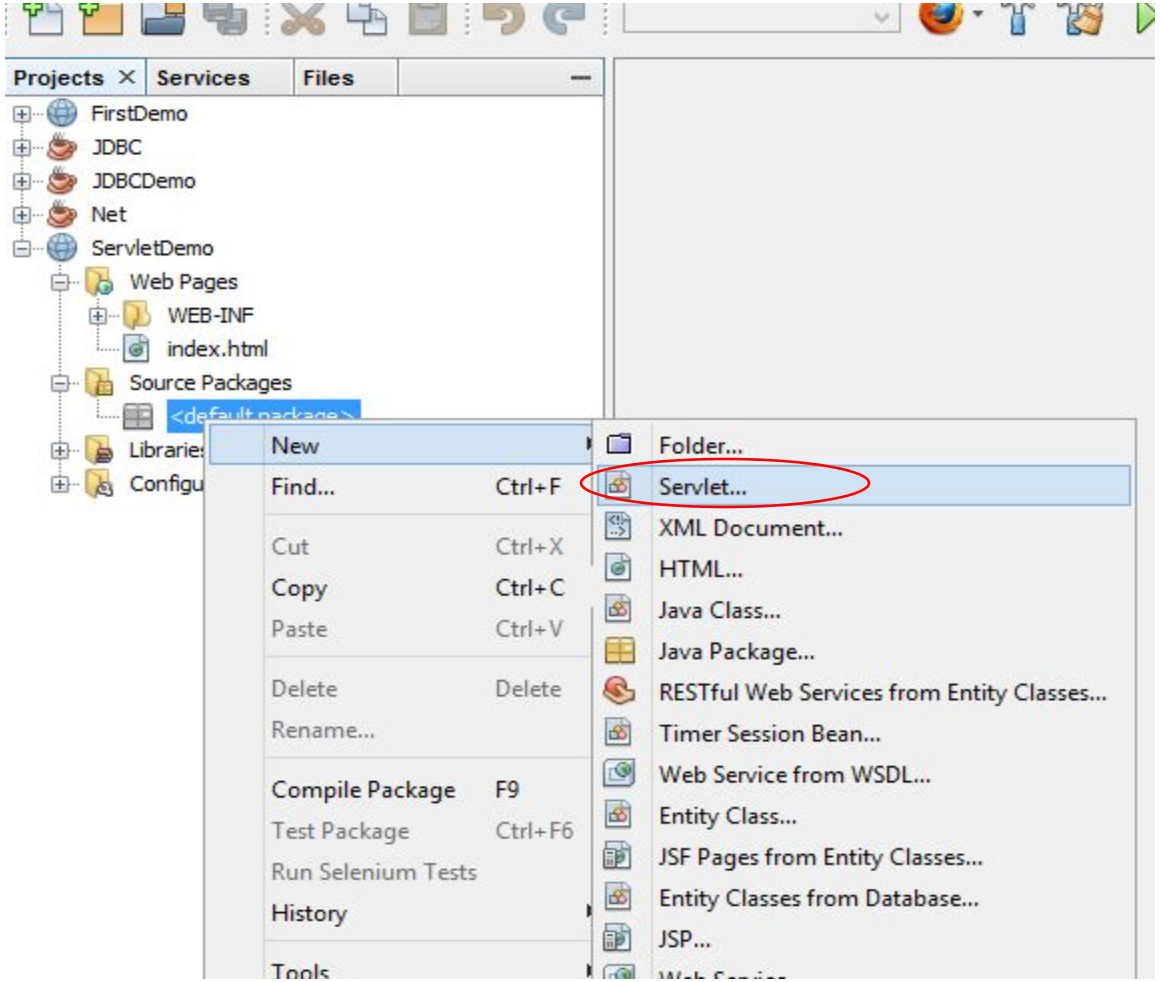
Steps for Servlet Program

- Step 5: The complete directory structure required for the Servlet Application will be created automatically by the IDE.



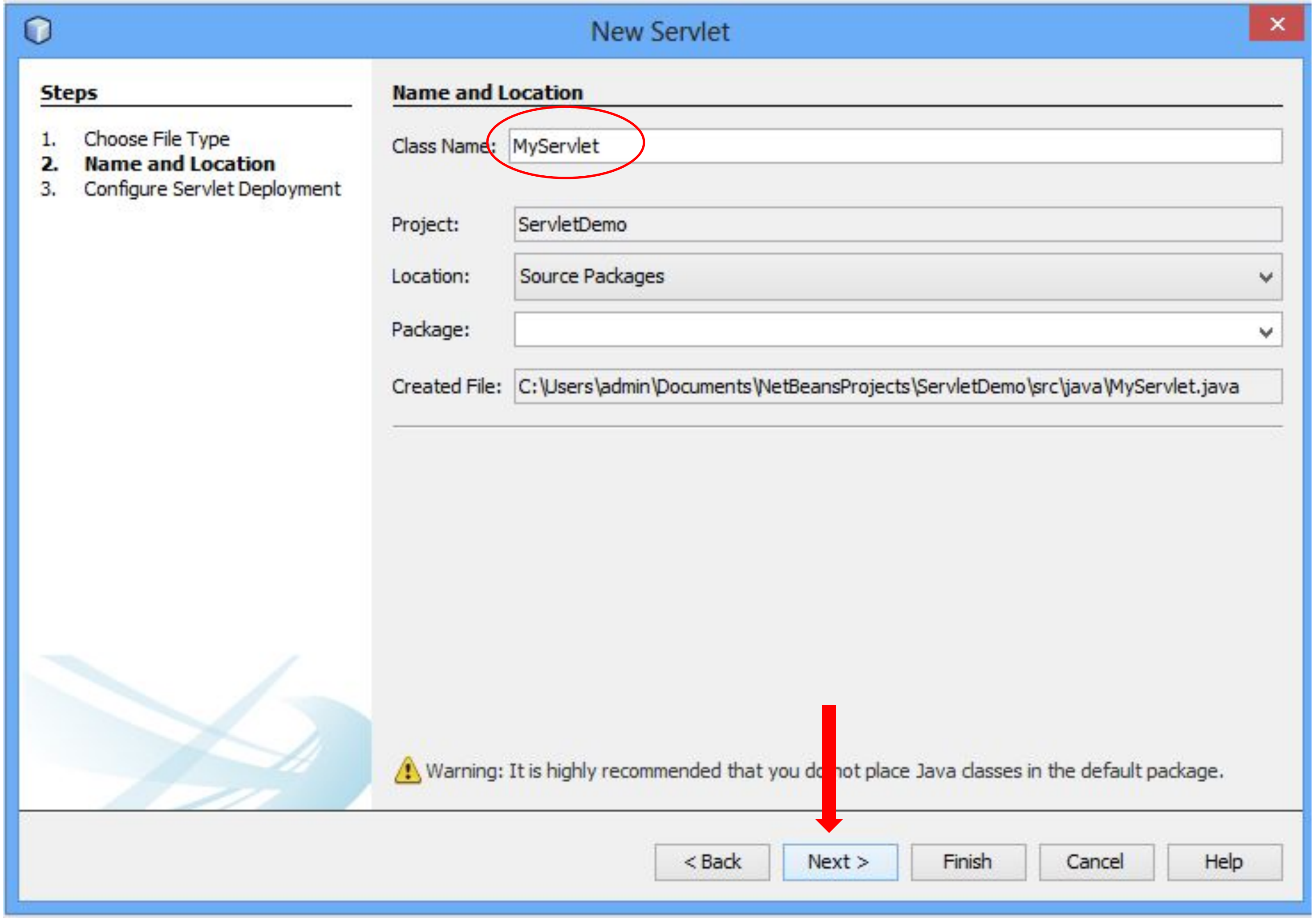
Steps for Servlet Program

- Step 6: To create a Servlet, open Source Package, right click on default packages -> New -> Servlet.



Steps for Servlet Program

- Step 7: Give a Name to your Servlet class file



Class Name: MyServlet

Project: ServletDemo

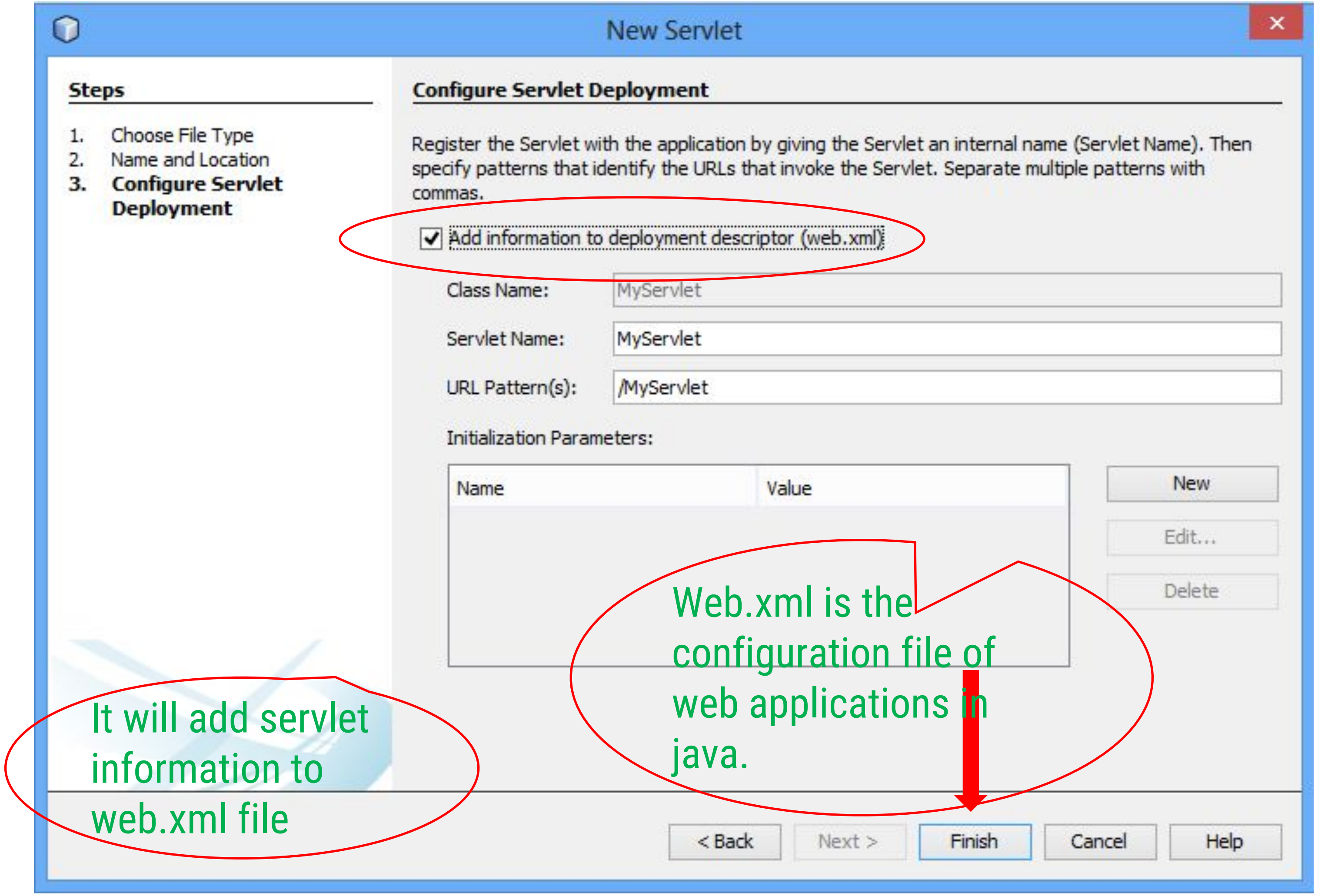
Location: Source Packages

Package:

Created File: C:\Users\admin\Documents\NetBeansProjects\ServletDemo\src\java\MyServlet.java

⚠ Warning: It is highly recommended that you do not place Java classes in the default package.

Steps for Servlet Program



Steps

1. Choose File Type
2. Name and Location
3. **Configure Servlet Deployment**

Configure Servlet Deployment

Register the Servlet with the application by giving the Servlet an internal name (Servlet Name). Then specify patterns that identify the URLs that invoke the Servlet. Separate multiple patterns with commas.

☑ Add information to deployment descriptor (web.xml)

Class Name: MyServlet

Servlet Name: MyServlet

URL Pattern(s): /MyServlet

Initialization Parameters:

Name	Value

It will add servlet information to web.xml file

Web.xml is the configuration file of web applications in java.

Step 8: Write servlet code: MyServlet.java

MyServet1.java

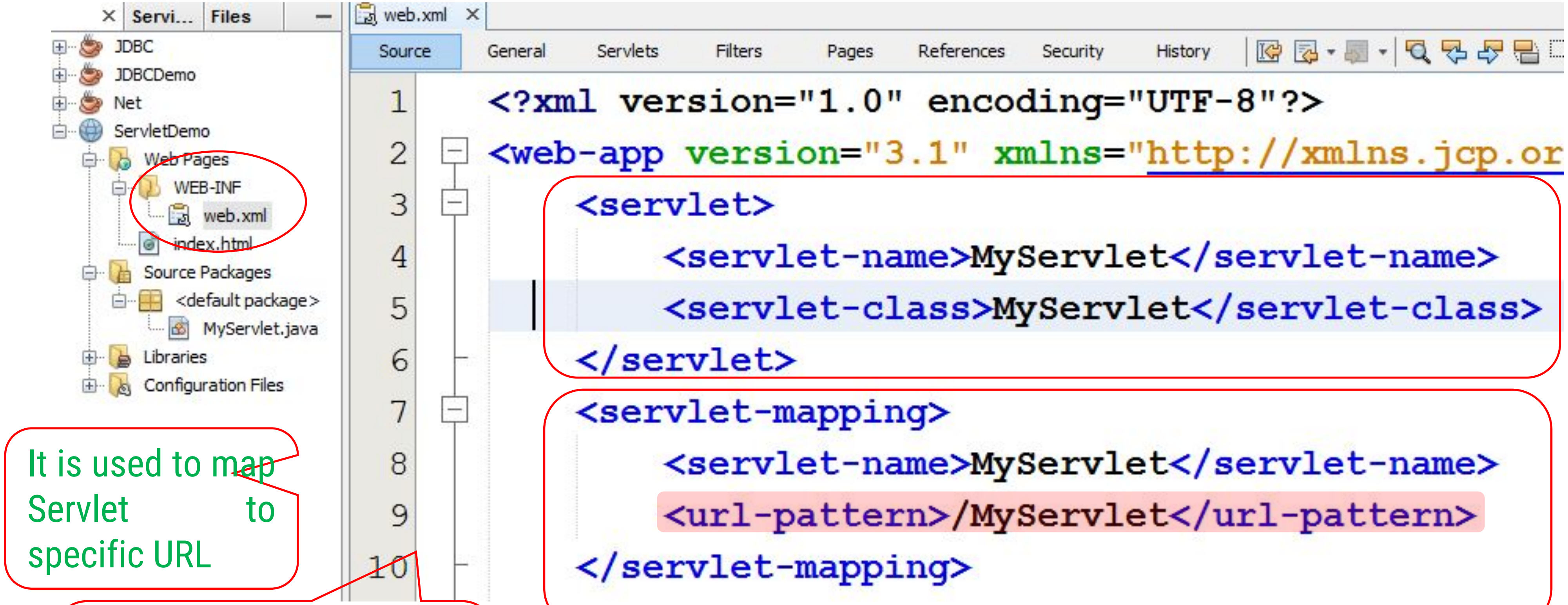
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class MyServlet1 extends HttpServlet
{    String msg="";
    PrintWriter out;
    public void init() throws ServletException
    {         msg="hello world: my first servlet program";    }
    public void doGet(HttpServletRequest request, HttpServletResponse response)throws     ServletException,IOException
    {
        response.setContentType("text/html");
        out=response.getWriter();
        out.println(msg);
    }
    public void destroy()
    {        out.close();
    }
}
```

MIME: Multipurpose Internet Mail Extensions

- A **MIME** type nomenclature includes a **type** and **subtype** separated by a forward slash.
- It is a **HTTP header** that provides the description about what are you sending to the browser.
    - text/html
    - text/plain
    - text/css
    - text/richtext
    - application/msword
    - application/jar
    - application/pdf
    - images/jpeg images/png images/gif
    - audio/mp3
    - video/mp4
- MIME is a standard set to Internet to notify the format of the file contents.

Steps for Servlet Program

- Step 9: open web.xml



```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.or
    <servlet>
        <servlet-name>MyServlet</servlet-name>
        <servlet-class>MyServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>MyServlet</servlet-name>
        <url-pattern>/MyServlet</url-pattern>
    </servlet-mapping>
```

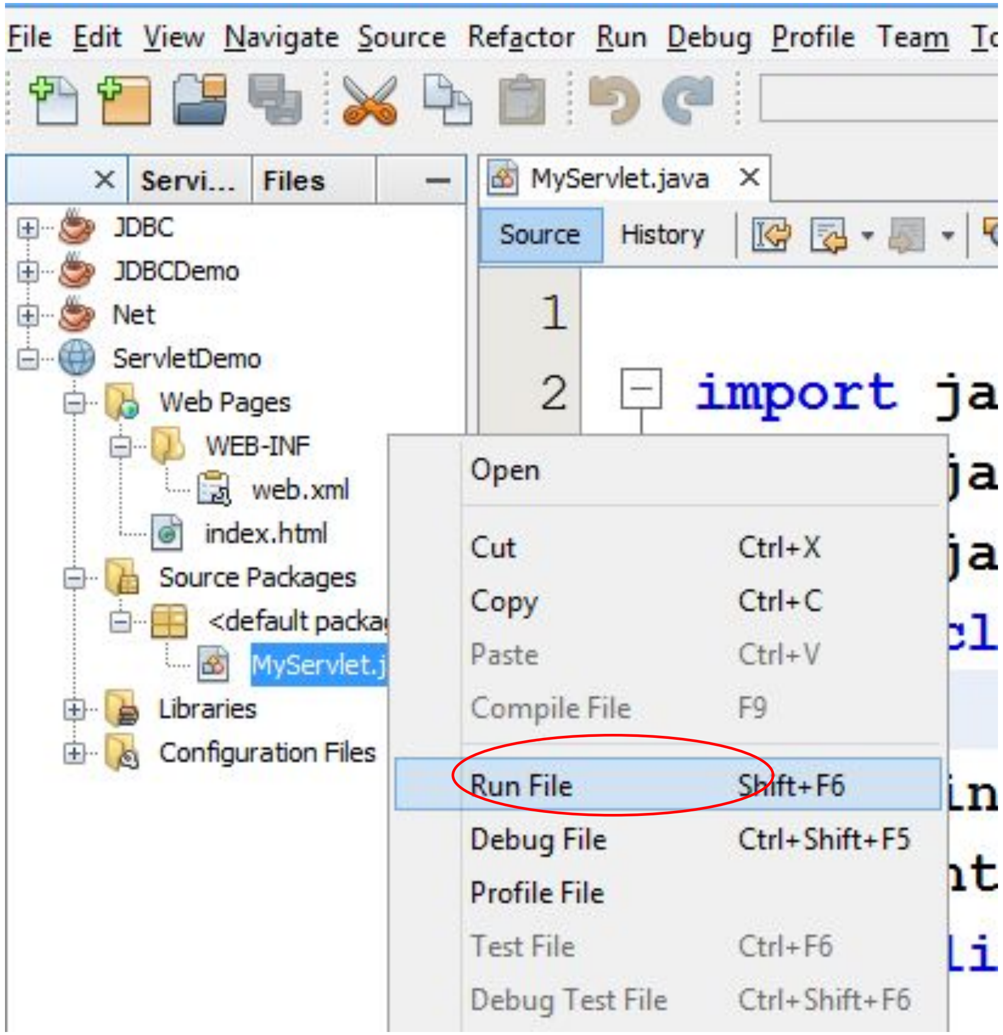
It is used to map Servlet to specific URL

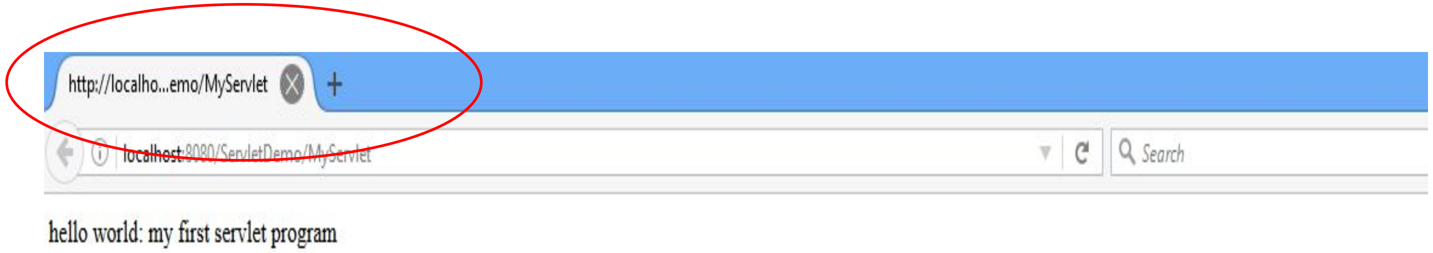
Configuration of servlet using <servlet>

Map the servlet to a URL. This can be done using <servlet-mapping> element.

Steps for Servlet Program

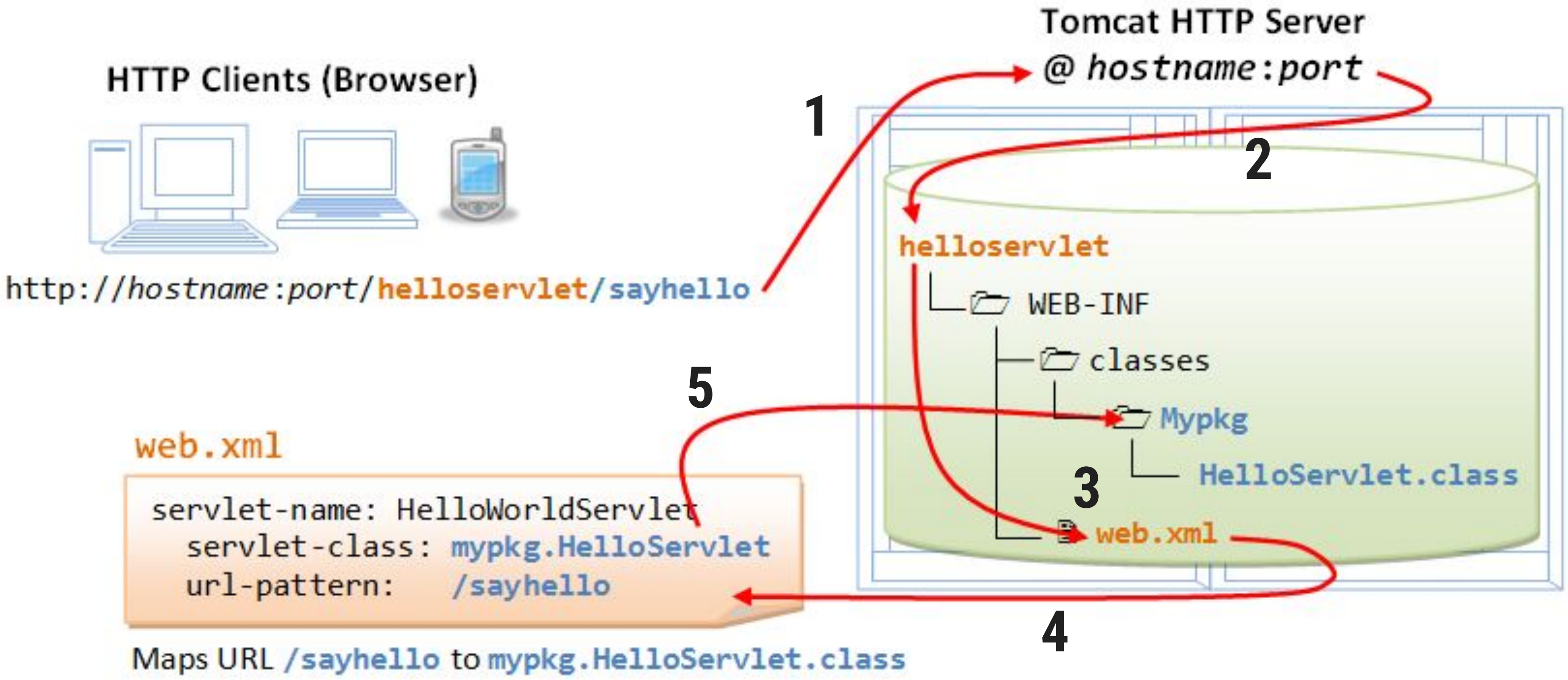
- Step 11: Run your application, right click on your Project and select **Run**




http://localho...emo/MyServlet
localhost:8080/ServletDemo/MyServlet
Search

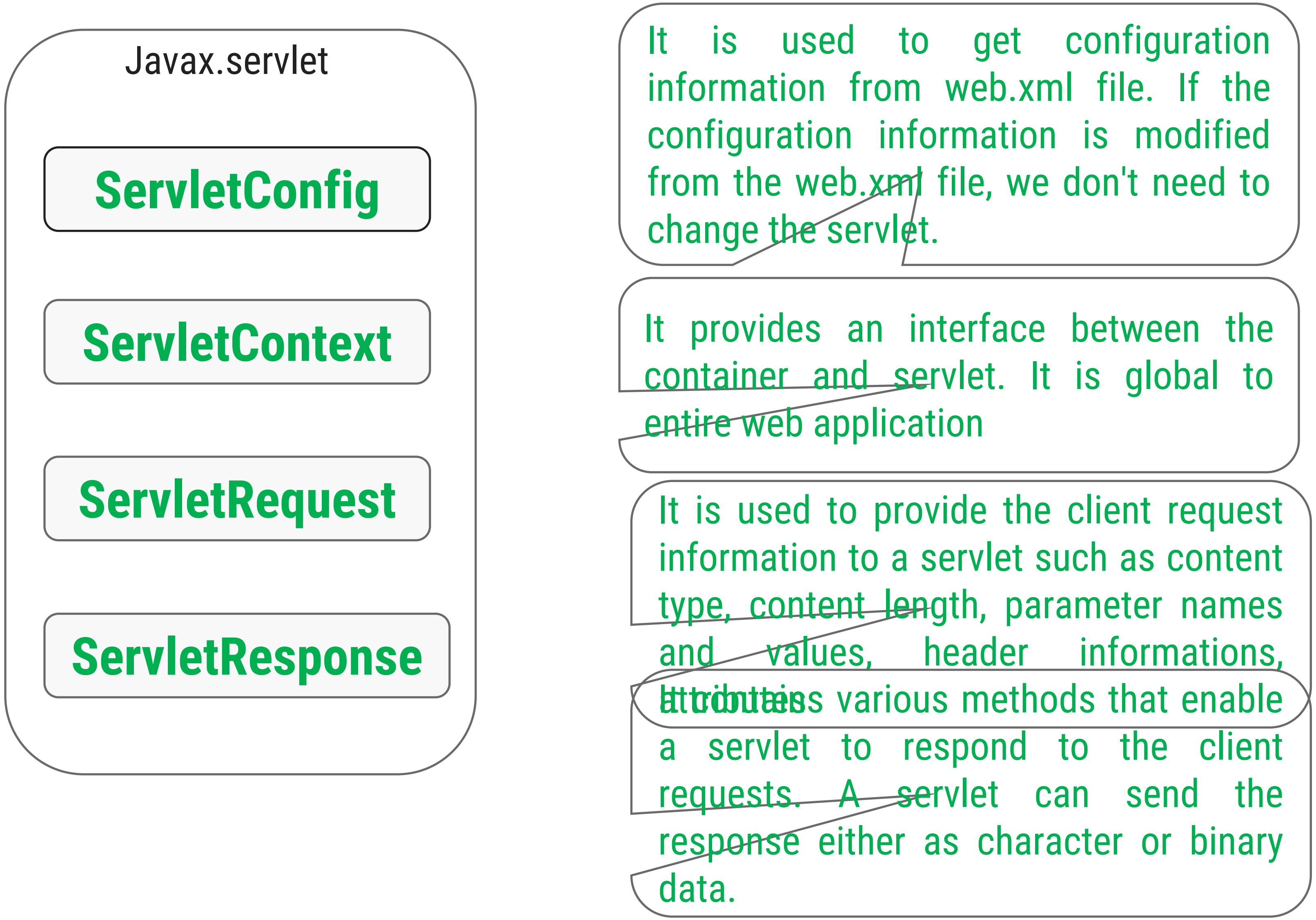
```
hello world: my first servlet program
```

Java Servlet


HTTP Clients (Browser)
http://hostname:port/helloservlet/sayhello
Tomcat HTTP Server
@ hostname:port
1
2
helloservlet
WEB-INF
classes
Mypkg
HelloServlet.class
3
web.xml
4
5
web.xml
servlet-name: HelloWorldServlet
servlet-class: mypkg.HelloServlet
url-pattern: /sayhello
Maps URL /sayhello to mypkg.HelloServlet.class

Ref: https://www.ntu.edu.sg/home/ehchua/programming/java/JavaServlets.html

javax.servlet Interface



Javax.servlet

ServletConfig

ServletContext

ServletRequest

ServletResponse

It is used to get configuration information from web.xml file. If the configuration information is modified from the web.xml file, we don't need to change the servlet.

It provides an interface between the container and servlet. It is global to entire web application

It is used to provide the client request information to a servlet such as content type, content length, parameter names and values, header informations, attributes

It contains various methods that enable a servlet to respond to the client requests. A servlet can send the response either as character or binary data.

Types of Servlet

- **Generic Servlet**
    - javax.servlet (package)
    - extends javax.servlet.Servlet
    - service method
    - service(ServletRequest req, ServletResponse res)
- **Http Servlet**
    - javax.servlet.http (package)
    - extends javax.servlet.HttpServlet
    - doGet(), doPost()
    - doGet(HttpServletRequest req,HttpServletResponse res)
    - doPost(HttpServletRequest req,HttpServletResponse res)

GenericServlet vs HttpServlet

GenericServlet	HttpServlet
javax.servlet.GenericServlet	javax.servlet.http.HttpServlet
It defines a generic, protocol-independent servlet.	It defines a HTTP protocol specific servlet.
GenericServlet is a super class of HttpServlet class.	HttpServlet is a sub class of GenericServlet class.
Can handle all types of protocols	only HTTP specific protocols.
It supports only one abstract method:service()	It support doGet(), doPost() etc.

Deployment Descriptor

- Located @ WEB-INF directory
- File known as web.xml
- It controls the behavior of Java Servlet
- What does it contain?
    - XML Header
    - DOCTYPE
    - Web-app element
    - The Web-app element should contain a servlet element with 3 sub-element.
        - <servlet-name>: name used to access java servlet
        - <servlet-class>: class name of java servlet
        - <init-param>: for initialization parameter

Deployment Descriptor: web.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE web-app

    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

    <servlet>

        <servlet-name>MyServlet</servlet-name>

        <servlet-class>MyServlet</servlet-class>

        <init-param>

            <param-name>name</param-name>

            <param-value>cxcy</param-value>

        </init-param>

    </servlet>
```

xml header

Document Type Definition

Name used to access Java Servlet

Name of servlet .java class

Used to pass parameters to a servlet from the web.xml file.

map the servlet to a URL or URL pattern

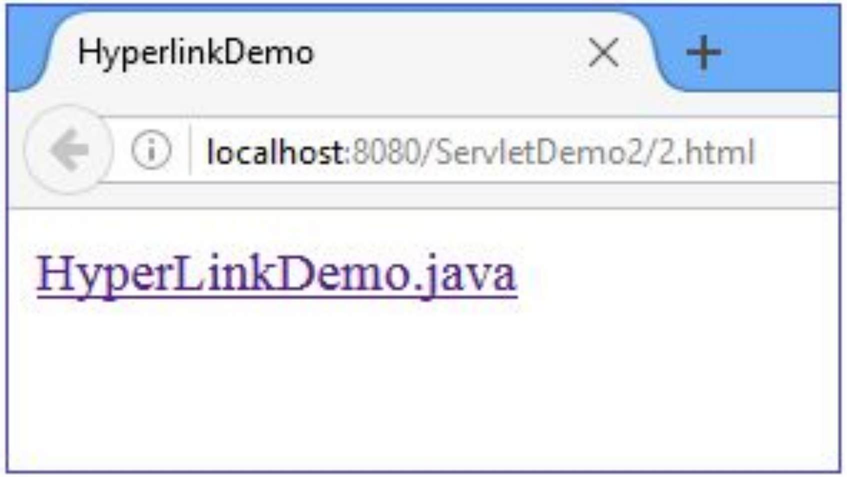
Controls behavior of Servlet

Program to call servlet from html file

- Write a java Servlet program to call servlet from html hyperlink.

2.html

```
<html>
 <head>
  <title> HyperLinkDemo </title>
 </head>
 <body>
   <a href ="/ServletDemo2/HyperLinkDemo">HyperLinkDemo.java </a>
 </body>
</html>
```



HyperlinkDemo

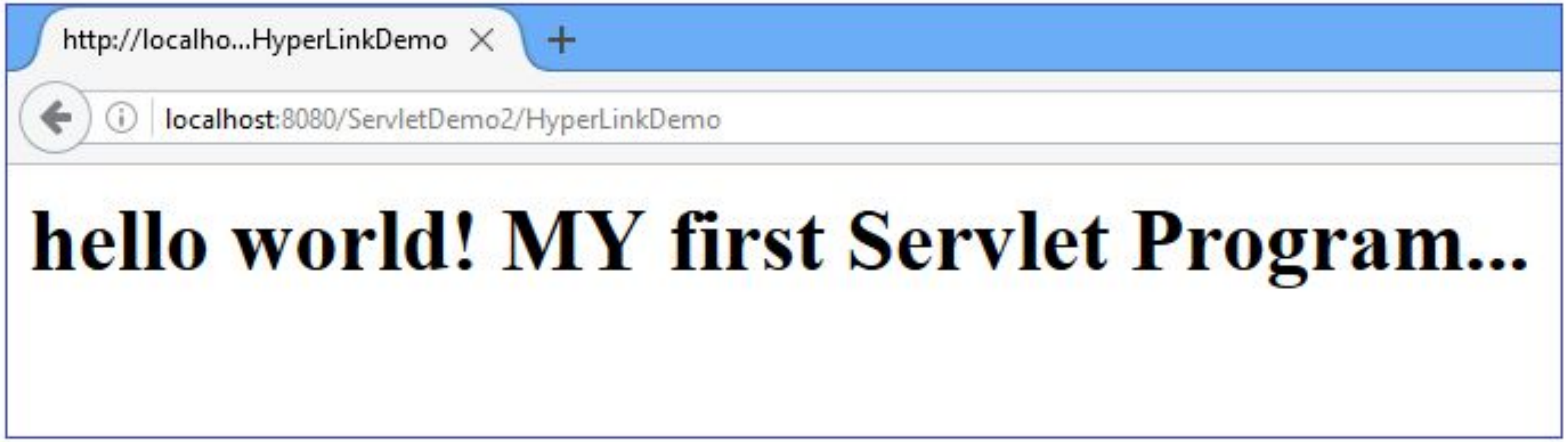
localhost:8080/ServletDemo2/2.html

HyperLinkDemo.java

Servlet Program: HyperLinkDemo.java

HyperLinkDemo.java

```
import javax.servlet.*
import javax.servlet.h
import java.io.*;
public class HyperLink
{   String msg="";
    PrintWriter out;
    public void init(S
    {   msg="hello wor
    }
    public void doGet(HttpServletRequest request,HttpServletResponse
                 response) throws ServletException,IOException
    {   response.setContentType("text/html");
        out=response.getWriter();
        out.println("<h1>"+msg+"</h1>");
    }
    public void destroy()
    {   out.close();
    }
}
```



http://localho...HyperLinkDemo

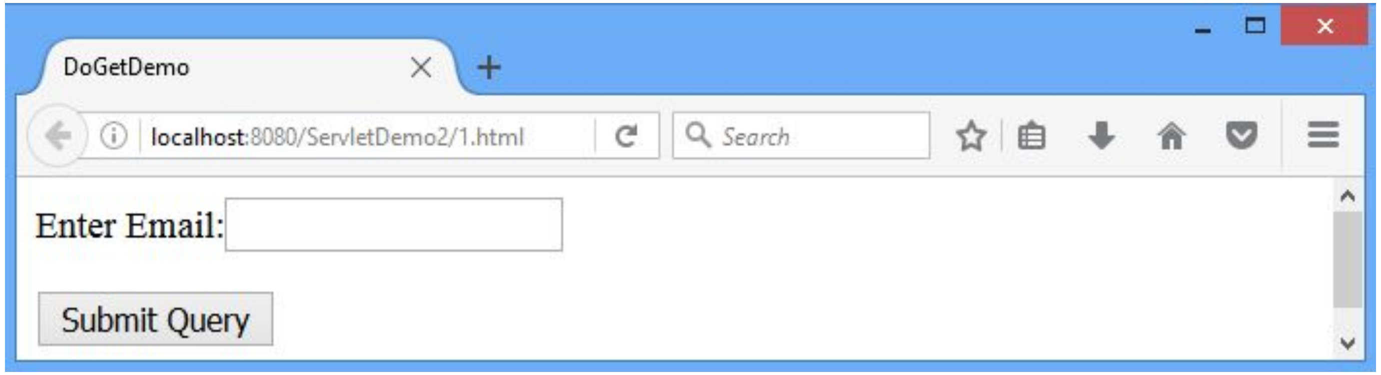
localhost:8080/ServletDemo2/HyperLinkDemo

hello world! MY first Servlet Program...

doGet()

1.html

```html
<html>
    <head>
        <title> DoGetDemo </title>
</head>
    <body>
        <form action="/ServletDemo2/DoGetDemo">
            Enter Email:<input type="text" name="email">
            <p><input type="submit"></p>
        </form>
    </body>
</html>
```



doGet()

DoGetDemo.java

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class DoGetDemo extends HttpServlet
{    PrintWriter out;
    public void init(ServletConfig config)throws ServletException
    {
    }
    public void doGet(HttpServletRequest request,HttpServletResponse response)throws
ServletException,IOException
    {
        String email=request.getParameter("email");
        response.setContentType("text/html");
        out =response.getWriter();
        out.println("my email:"+email);
    }
    public void destroy()
    {        out.close();
    }
 }
```

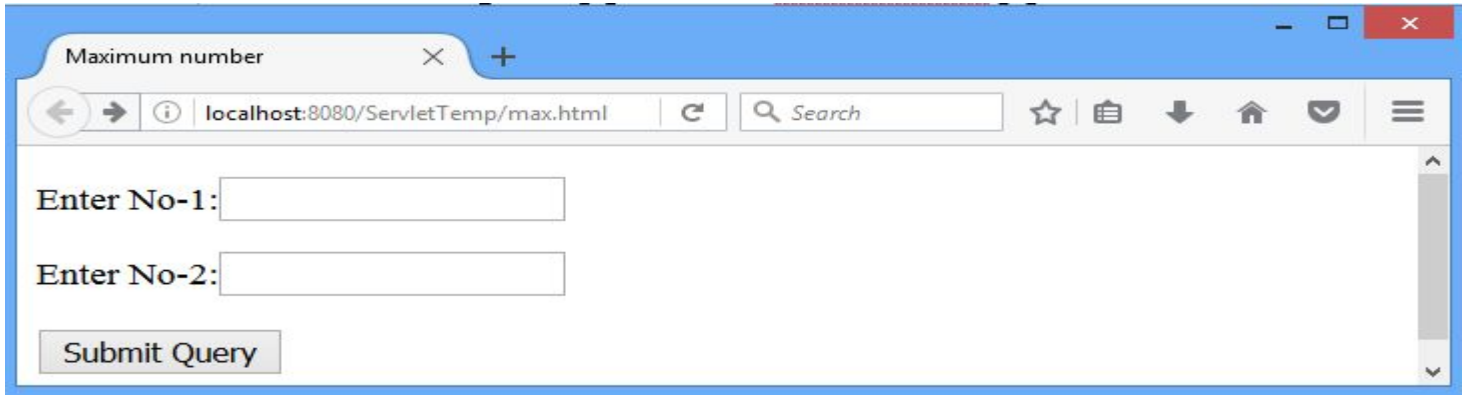
String **getParameter**(String name)
Returns the value of a request parameter as a String

doPost()

- Write a Servlet program to enter two numbers and find maximum among them.

max.html

```
<!DOCTYPE html>
<html>
    <head>
        <title> Maximum number </title>
</head>
    <body>
        <form action="/ServletTemp/Max" method="POST" >
            <p>Enter No-1:<input type="text" name="no1"></p>
            <p>Enter No-2:<input type="text" name="no2"></p>
            <p><input type="submit"></p>
        </form>
    </body>
</html>
```



Enter No-1:

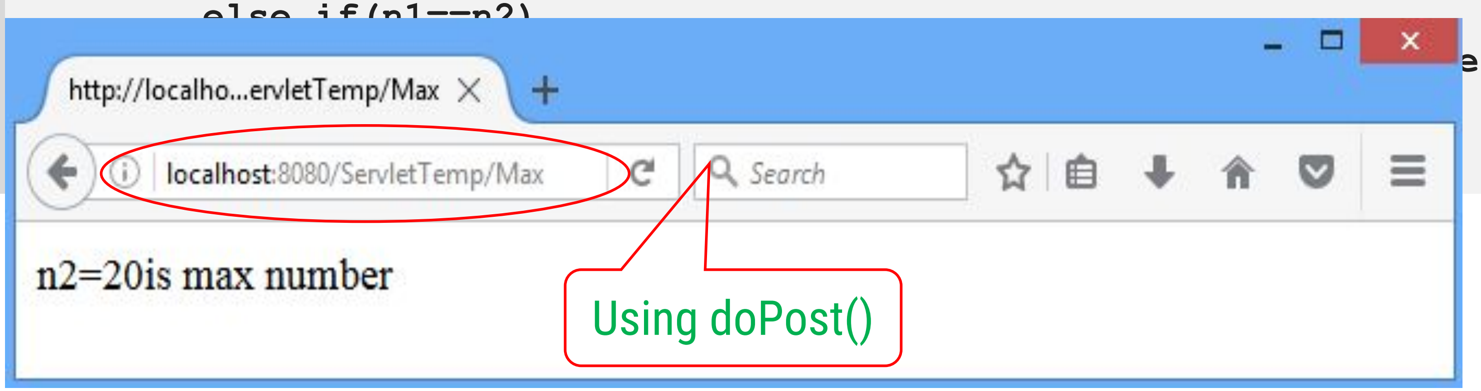
Enter No-2:

Submit Query

doPost()

Max.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Max extends HttpServlet
{
     public void doPost(HttpServletRequest request, HttpServletResponse response)throws
    ServletException,IOException
    {   int n1=0,n2=0;
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        n1=Integer.parseInt(request.getParameter("no1"));
        n2=Integer.parseInt(request.getParameter("no2"));
        if(n1>n2)
            out.println("n1="+n1+"is max number");
        else if(n2>n1)
            out.println("n2="+n2+"is max number");
        else if(n1==n2)
                                                               ers");
```


http://localho...ervletTemp/Max
localhost:8080/ServletTemp/Max
Search

```
n2=20is max number
```

Using doPost()

ServletConfig Interface

- It is used to get configuration information from web.xml file.

- If the configuration information is modified from the web.xml file, we don't need to change the servlet.

Method :

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

Example

```
String str = config.getInitParameter("name")
```

web.xml

```
<init-param>
<param-name>name</param-name>
```

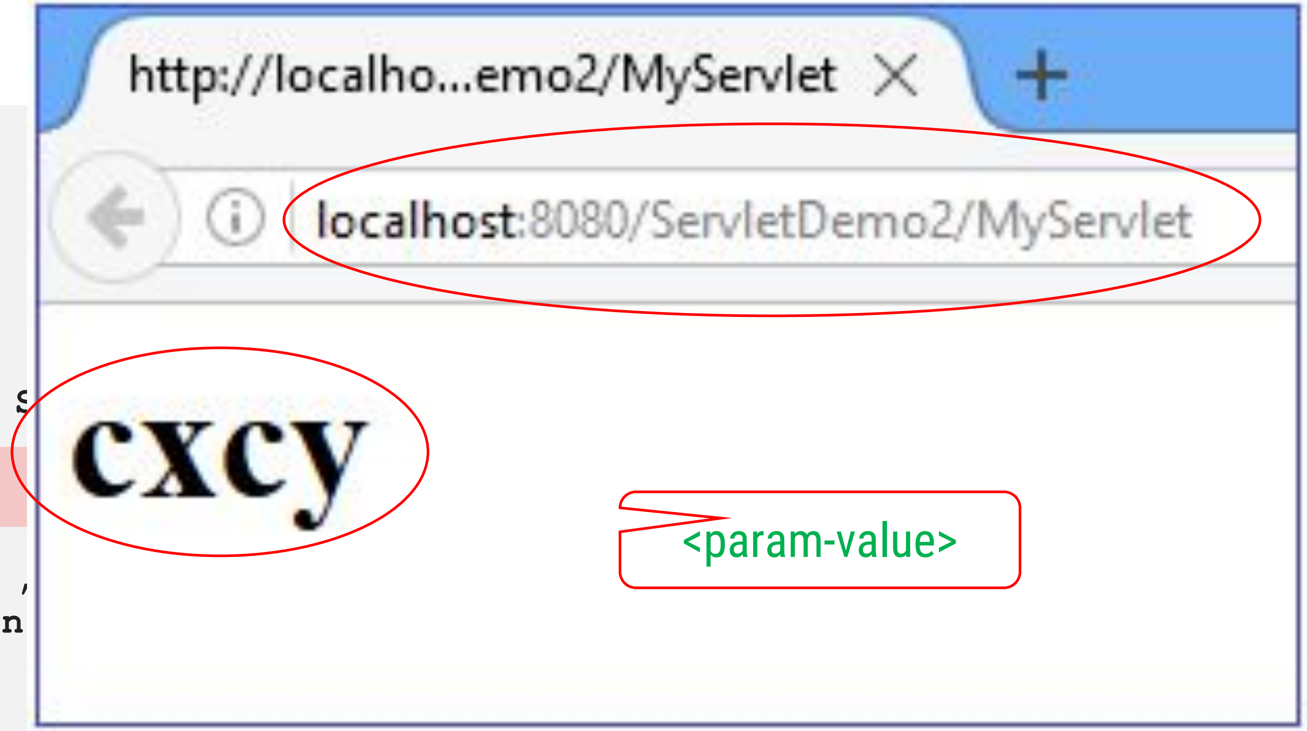
Servlet Config: web.xml

```
<web-app>
    <servlet>
        <servlet-name>MyServlet</servlet-name>

<servlet-class>MyServlet</servlet-class>
        <init-param>
            <param-name>name</param-name>
            <param-value>cxcy</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>MyServlet</servlet-name>
        <url-pattern>/MyServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

Servlet Config: MyServlet.java

MyServlet.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class MyServlet extends HttpServlet
{    String msg;
     PrintWriter out;
     public void init(ServletConfig config)throws S
     {
      msg = config.getInitParameter("name");
     }
     public void doGet(HttpServletRequest request ,
                         ServletException,IOException
     {    response.setContentType("text/html");
          out = response.getWriter();
          out.println("<h1>"+ msg +"</h1>");
     }
     public void destroy()
     {         out.close();
     }
}
```



localhost:8080/ServletDemo2/MyServlet

cxcy

<param-value>

ServletContext Interface

- ServletContext is created by the **web container** at time of deploying the project.
- It can be used to get configuration information from web.xml file.
- There is only one ServletContext object per web application.
- If any information is shared to many servlet, it is better to provide it from the web.xml file using the <context-param> element.

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

used to define initialization parameter in the application scope.

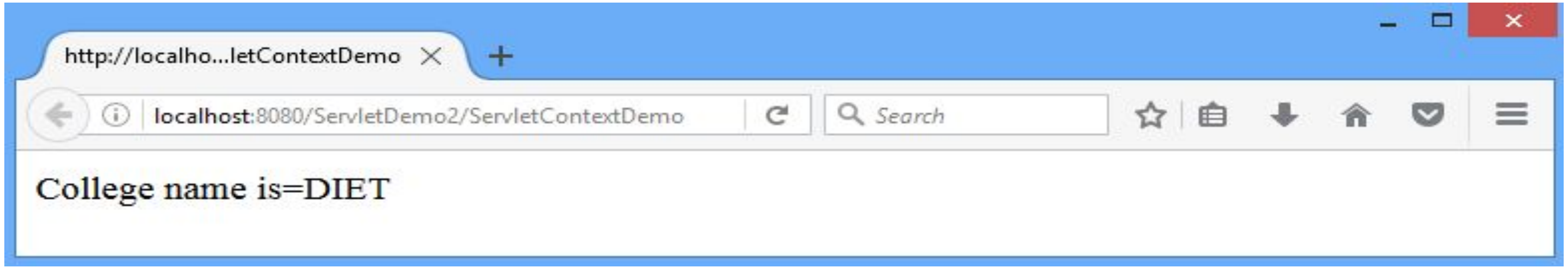
web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
    <servlet>
        <servlet-name>ServletContextDemo</servlet-name>
        <servlet-class>ServletContextDemo</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ServletContextDemo</servlet-name>
        <url-pattern>/ServletContextDemo</url-pattern>
    </servlet-mapping>
    <context-param>
        <param-name>name</param-name>
        <param-value>DIET</param-value>
    </context-param>
</web-app>
```

ServletContextDemo.java

ServletContextDemo.java

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletContextDemo extends HttpServlet
{
    public void doGet(HttpServletRequest req,HttpServletResponse res)  throws
                         ServletException,IOException
   {   res.setContentType("text/html");
       PrintWriter out=res.getWriter();
       //creating ServletContext object
       ServletContext context=getServletContext();
       //Getting the value of the initialization parameter and printing it
       String college=context.getInitParameter("name");
       out.println("College name is="+college);
       out.close();
   }
}
```


http://localho...letContextDemo
localhost:8080/ServletDemo2/ServletContextDemo
Search

College name is=DIET

Servlet Config vs Servlet Context

Servlet Config	Servlet Context
ServletConfig object is one per servlet class	ServletContext object is global to entire web application
Object of ServletConfig will be created during initialization process of the servlet	Object of ServletContext will be created at the time of web application deployment
Scope: As long as a servlet is executing, ServletConfig object will be available, it will be destroyed once the servlet execution is completed.	***Scope:*** As long as web application is executing, ServletContext object will be available, and it will be destroyed once the application is removed from the server.
We should give request explicitly, in order to create ServletConfig object for the first time	ServletContext object will be available even before giving the first request
In web.xml – <*init-param*> tag will be appear under <*servlet-class*> tag	In web.xml – <*context-param*> tag will be appear under <*web-app*> tag

HttpServletRequest: Methods

String **getContextPath**()	Returns the portion of the request URI that indicates the context of the request.
Enumeration **getHeaderNames**()	Returns an enumeration of all the header names this request contains.
String **getHeader**(String name)	Returns the value of the specified request header as a String.
String **getQueryString**()	Returns the query string that is contained in the request URL after the path.
String **getServletPath**()	Returns the part of this request's URL that calls the servlet. This path starts with a "/" character and includes either the servlet name or a path to the servlet
String **getMethod**()	Returns the name of the HTTP method with which this request was made, for example GET or POST

HttpServletRequest: Methods

String **getContextPath**()	Returns the portion of the request URI that indicates the context of the request.

getContextPath

```java
public void doGet(HttpServletRequest request, HttpServletResponse response)
{
    out.println("<p>request.getContextPath():" +request.getContextPath()+"</p>");
}
```

Output

```
request.getContextPath():/ServletTemp
```

HttpServletRequest: Methods

Enumeration **getHeaderNames**()	Returns an enumeration of all the header names this request contains.

getHeaderNames

```
public void doGet(HttpServletRequest request,HttpServletResponse response)
{
       Enumeration h=request.getHeaderNames();
       while(h.hasMoreElements())
     {
       String paramName = (String)h.nextElement();
       out.print("<p>" + paramName + "\t");
       String paramValue = request.getHeader(paramName);
       out.println( paramValue + "</p>\n");
     }
}
```

Output

```
host     localhost:8080
user-agent Mozilla/5.0 (Windows NT 6.2; WOW64;rv:50.0) Gecko/20100101 Firefox/50.0
accept   text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
accept-language en-US,en;q=0.5
accept-encoding gzip, deflate
connection keep-alive
upgrade-insecure-requests 1
```

HttpServletRequest: Methods

String **getHeader**(String name)	Returns the value of the specified request header as a String.

getHeader

```
public void doGet(HttpServletRequest request,HttpServletResponse response)
{
    out.println("<p>request.getHeader(): "  +request.getHeader("host")+"</p>");
        out.println("<p>request.getHeader(): "  +request.getHeader("referer")+"</p>");
}
```

Output

```
request.getHeader():host=localhost:8080
request.getHeader():referer=http://localhost:8080/ServletTemp/servletmeth.html
```

HttpServletRequest: Methods

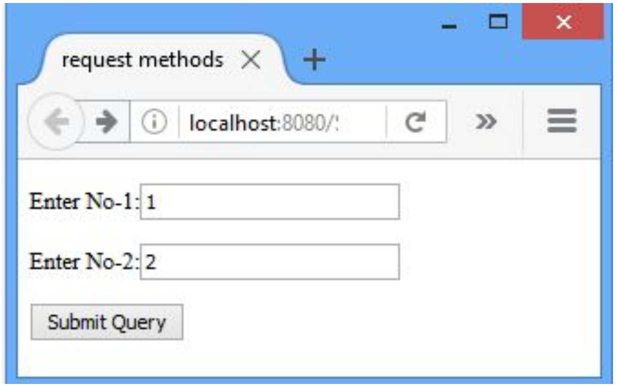
String **getQueryString**()	Returns the query string that is contained in the request URL after the path.

getQueryString

```
public void doGet(HttpServletRequest request,HttpServletResponse response)
{
    out.println("<p>request.getQueryString():" +request.getQueryString()+"</p>");

}
```

Output

```
requrest.getQueryString(): no1=1&no2=2
```



Enter No-1: 1

Enter No-2: 2

HttpServletRequest: Methods

String **getServletPath**()	Returns the part of this request's URL that calls the servlet. This path starts with a "/" character and includes either the servlet name or a path to the servlet

getServletPath

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
{
    out.println("<p>request.getServletPath():"  +request.getServletPath()+"</p>");
}
```

Output

```
request.getServletPath(): /ServletMeth
```

HttpServletRequest: Methods

String **getMethod**()	Returns the name of the HTTP method with which this request was made, for example GET or POST

getServletPath

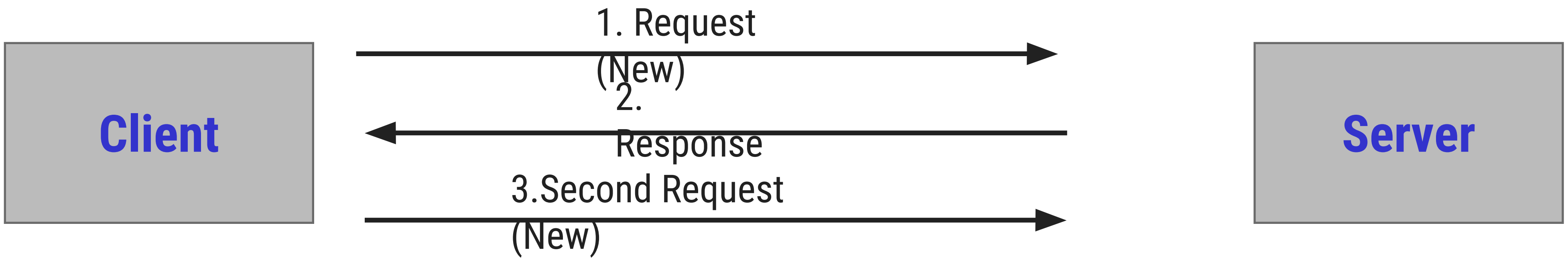
```
public void doGet(HttpServletRequest request, HttpServletResponse response)
{
    out.println("<p>request.getMethod():"+request.getMethod()+"</p>");
}
```

Output

```
request.getMethod(): GET
```

Session Management in Servlets

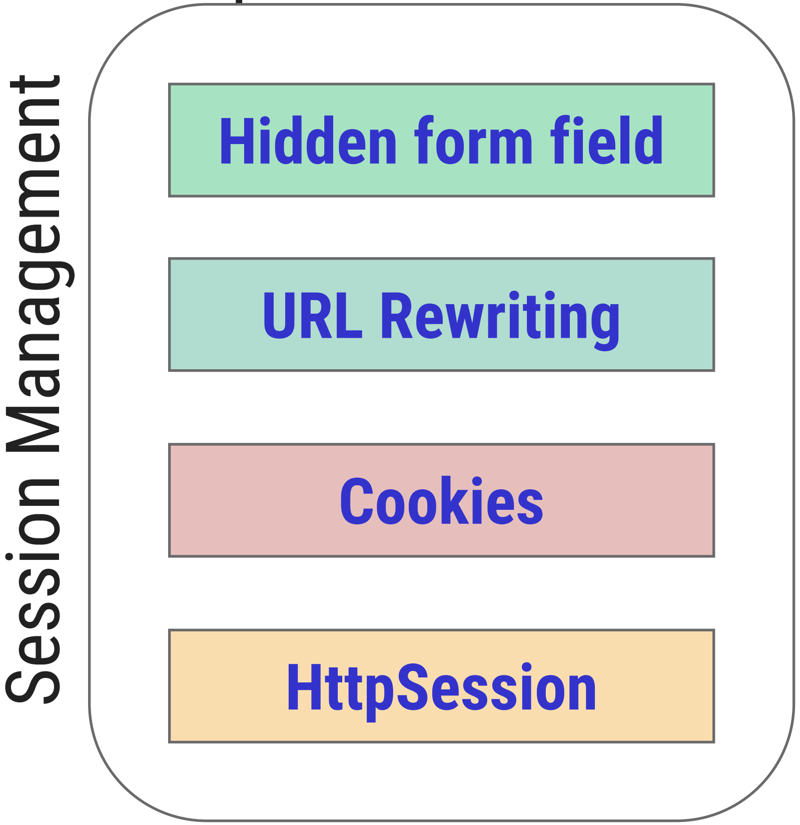
- A session refers to the entire interaction between a client and a server from the time of the client's first request, which generally begins the session, to the time of last request/response.
- **Why we require Session?**
  - HTTP is a "stateless" protocol which means each time a client retrieves a Web page, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.
  - Session is required to keep track of users and their information.



- When a User logs into your website, no matter on which web page he visits after logging in, his credentials will be with the server, until user logs out.
- So this is managed by creating a session.

Session Management

- Session Management is a mechanism used by the **Web container** to store session information for a particular user.
- There are four different techniques for session management.



Session Management: Hidden form field

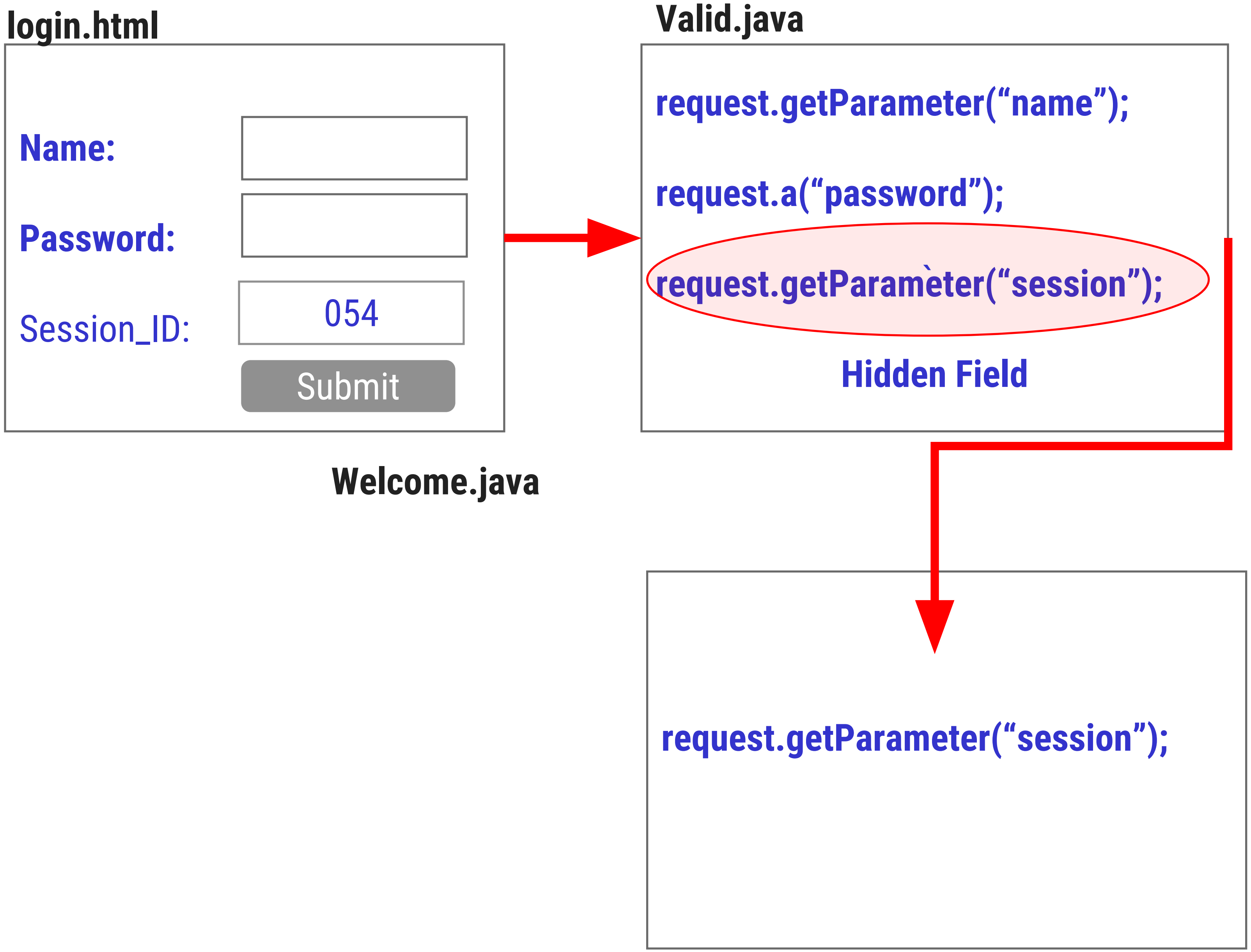
- Hidden Form Field, **a hidden (invisible) textfield** is used for maintaining the state of an user.

- In such case, we store the information in the hidden field and get it from another servlet.

Example

```
<input type="hidden" name="session_id" value="054">
```

Session Management: Hidden form field



login.html

Name:

Password:

Session_ID: 054

Submit

Valid.java

request.getParameter(“name”);

request.a(“password”);

request.getParamèter(“session”);

Hidden Field

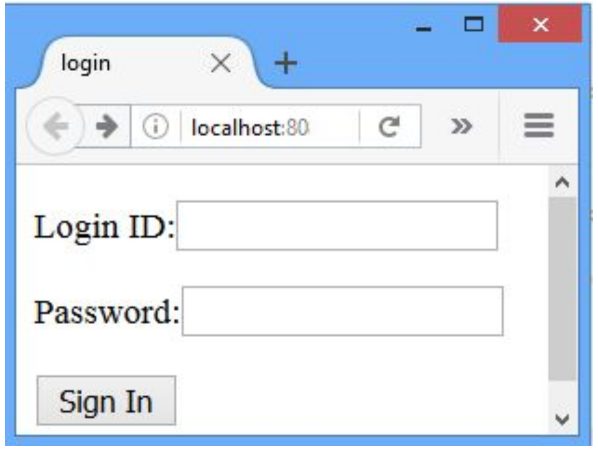
Welcome.java

request.getParameter(“session”);

Session Management: Hidden form field

login.html

```html
<html>
    <head>
        <title>login</title>
    </head>
 <body>
 <form action="/Session/Valid" method="POST">
    <p>Login ID:<input type="text" name="login"></p>
        <p>Password:<input type="text" name="pwd"></p>
    <p><input type="hidden" name="session_id" value="054"></p>
    <p><input type="submit" value="Sign In"></p>
  </form>
 </body>
</html>
```



Login ID:

Password:

Session Management: Hidden form field

Valid.java

```java
public class Valid extends HttpServlet
{   public void doPost(HttpServletRequest request, HttpServletResponse
    response)  throws ServletException,IOException
  {
    response.setContentType("text/html");
    PrintWriter out=response.getWriter();
    RequestDispatcher rd;
    String login=request.getParameter("login");
    String pwd=request.getParameter("pwd");
    String session=request.getParameter("session_id");
    if(login.equals("java") && pwd.equals("servlet"))
    {
        rd=request.getRequestDispatcher("Welcome");
        rd.forward(request, response);
    }//if
    else
    {
        out.println("<p><h1>Incorrect LoginId/Password </h1></p>");
        rd=request.getRequestDispatcher("/login.html");
        rd.include(request, response);
    }//else
 }
}
```

Hidden Field

Session Management: Hidden form field

Welcome.java

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Welcome extends HttpServlet
{   public void doPost(HttpServletRequest request, HttpServletResponse response)
                      throws ServletException,IOException
    {   response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        String session=request.getParameter("session_id");
        String username=request.getParameter("login");
        out.println("<h1>"+"id:"+session+"</h1>");
        out.println("<h3>"+"Welcome "+username+"</h3>");
    }
}

```

Session Management: Hidden form field

- **Real application of hidden form field**
  - It is widely used in comment form of a website.
  - In such case, we store page id or page name in the hidden field so that each page can be uniquely identified.

- **Advantage of Hidden Form Field**
  - Easy to implement
  - It will always work whether cookie is disabled or not.

- **Disadvantage of Hidden Form Field**
  - It is maintained at server side.
  - Extra form submission is required on each pages.
  - Only textual information can be used.
  - It does not support hyperlink submission.
  - Security
    - Hidden field will be visible with GET method
    - User might view page source and can view hidden field

Session Management: URL Rewriting

- In URL rewriting, a token or identifier is appended to the URL of the next Servlet or the next resource.
- We can send parameter name/value pairs using the following format:
- **`URL ? Name1 = value1 & name2 = value2 &…`**

A name and a value is separated using an equal (=) sign

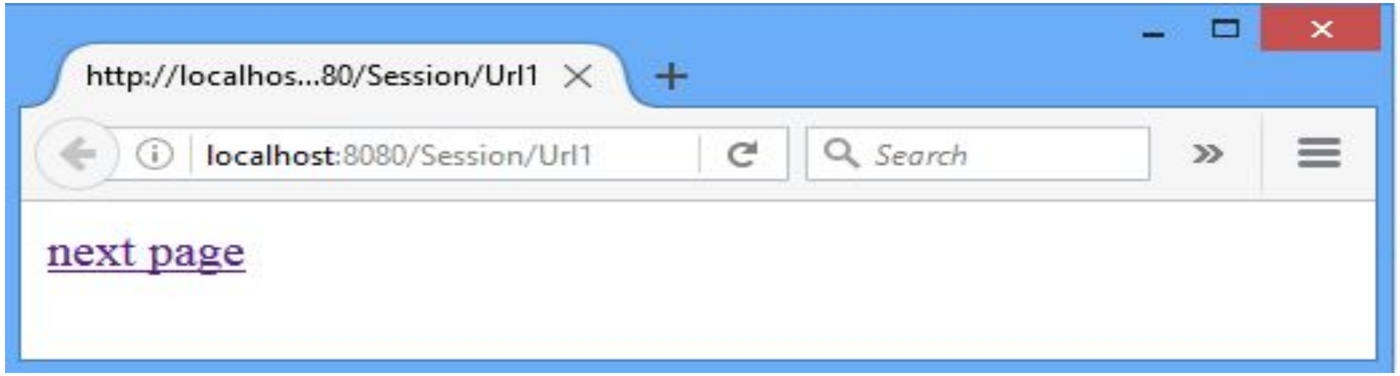
name/value pair is separated from another parameter using the ampersand(&)

- When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server.
- From a Servlet, we can use **getParameter()** method to obtain a parameter value.

Session Management: URL Rewriting

Url1.java

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Url1 extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse response)
                              throws ServletException,IOException
 {
   String url;
   response.setContentType("text/html");
   PrintWriter out=response.getWriter();
   //for URL rewriting
   url= "http://localhost:8080/Session/Url2?s_id1=054&s_id2=055";
   out.println("<a href="+url+">next page</a>");
 }
}
```

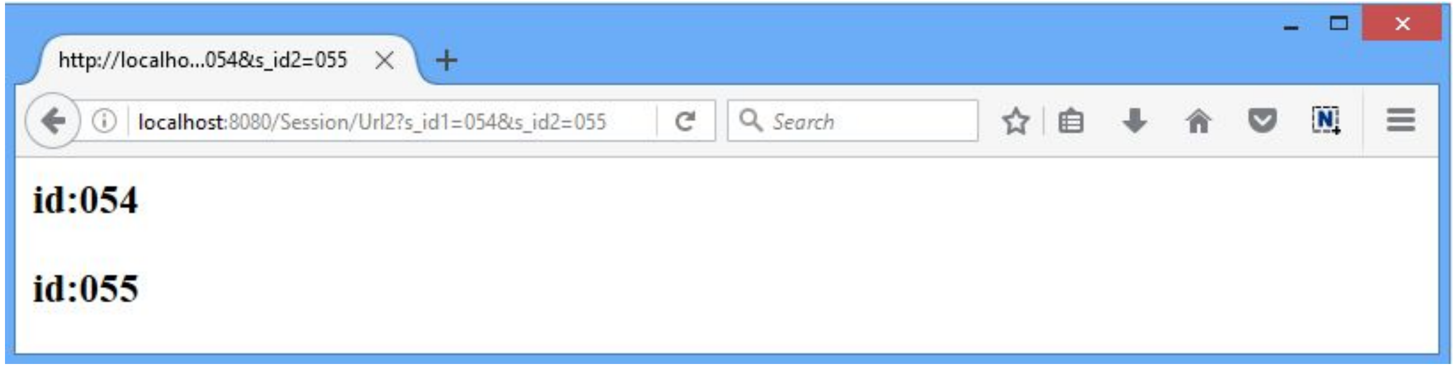

http://localhos...80/Session/Url1
localhost:8080/Session/Url1
Search

next page

Session Management: URL Rewriting

Url2.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Url2 extends HttpServlet
{   public void doGet(HttpServletRequest request, HttpServletResponse response)
                                  throws ServletException,IOException
    {   response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        String session1=request.getParameter("s_id1");
        String session2=request.getParameter("s_id2");
        out.println("<h3>"+"id:"+session1+"</h3>");
        out.println("<h3>"+"id:"+session2+"</h3>");
    }
}

```



localhost:8080/Session/Url2?s_id1=054&s_id2=055

```
id:054

id:055
```

Session Management: URL Rewriting

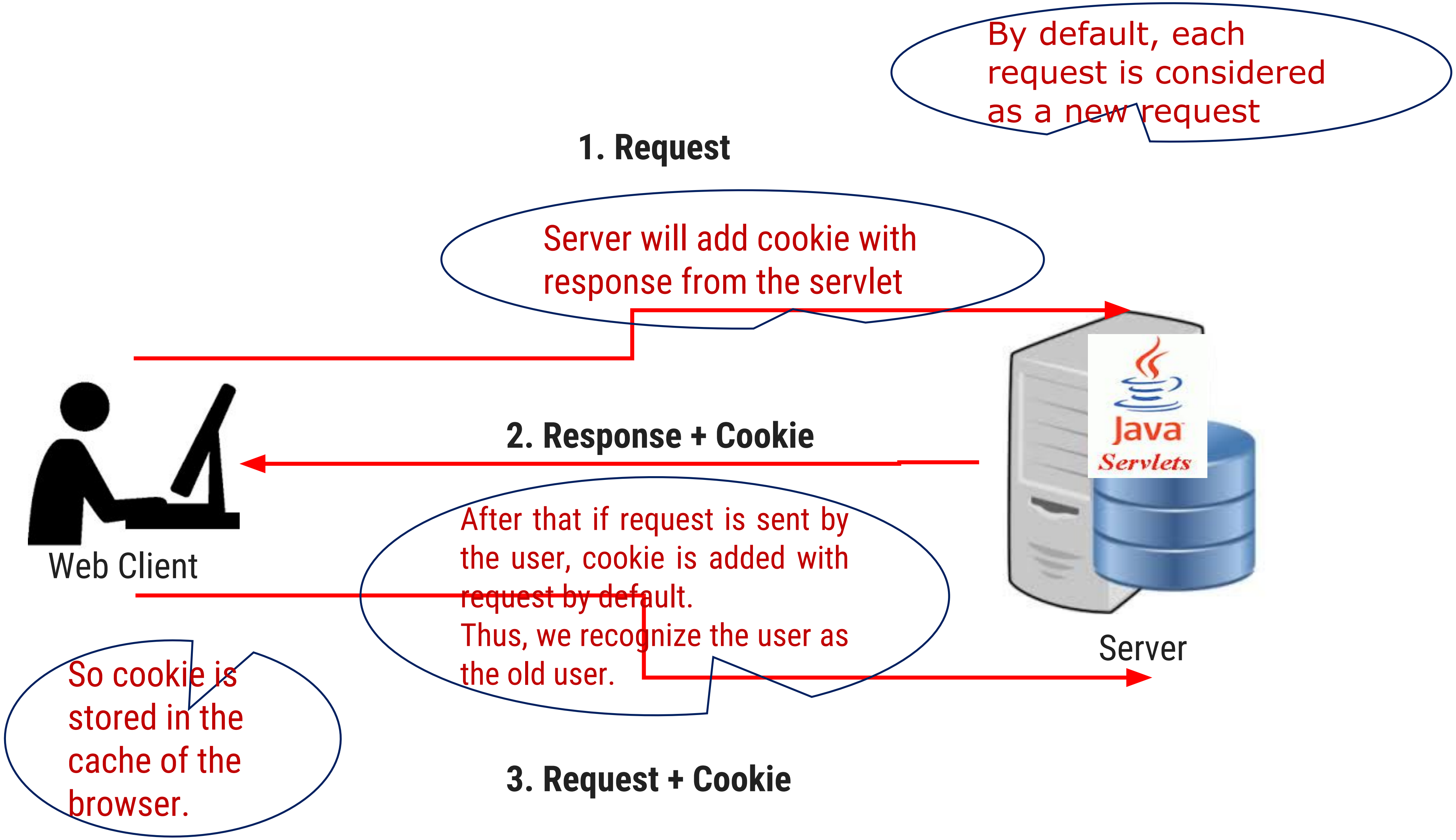
- **Advantage of URL Rewriting**
  - It will always work whether cookie is disabled or not (browser independent).
  - Extra form submission is not required on each pages.
- **Disadvantage of URL Rewriting**
  - It will work only with links.
  - It can send only textual information.
  - URL header size constraint.
  - Security
    - name/value field will be visible with URL followed by '?'.

Session Management: Cookies

- A **cookie** is a small piece of information that is persisted between the multiple client requests.

- A cookie has a
  - Name
  - Single value
  - Optional attributes such as
    - comment
    - path
    - domain qualifiers
    - a maximum age
    - version number

Session Management: Cookies



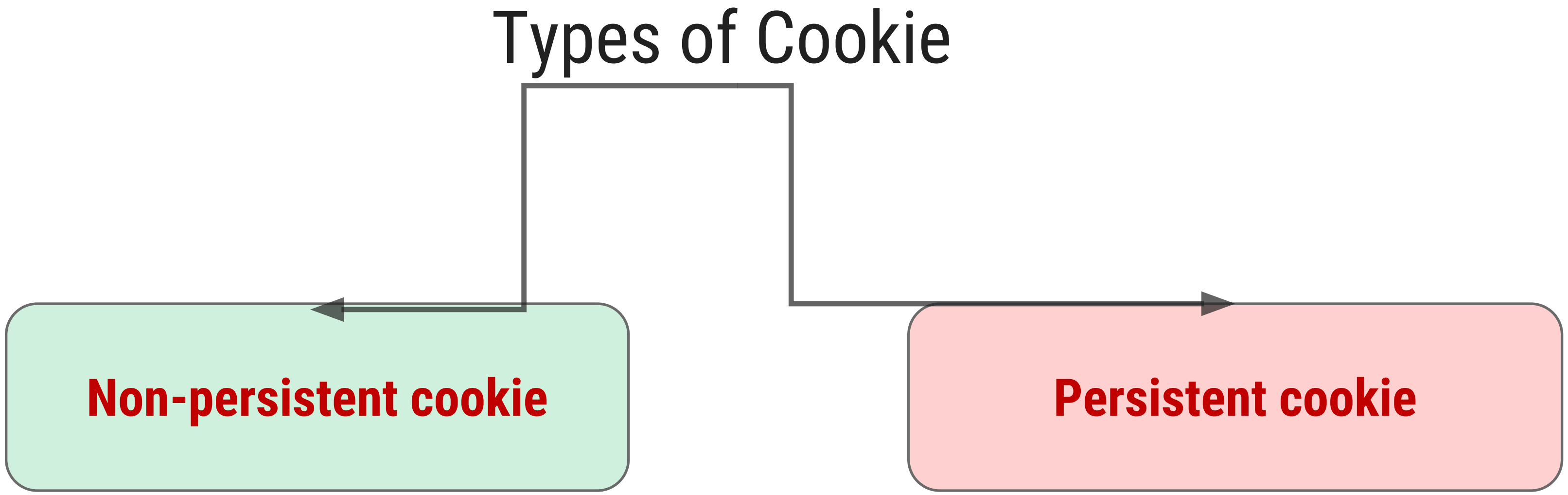
By default, each request is considered as a new request

Server will add cookie with response from the servlet

After that if request is sent by the user, cookie is added with request by default.
Thus, we recognize the user as the old user.

So cookie is stored in the cache of the browser.

Session Management: Cookies



Types of Cookie

Non-persistent cookie

Persistent cookie

- It is **valid for single session** only.
- It is removed each time when user closes the browser.

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

Session Management: Cookies

Cookie class

- **javax.servlet.http.Cookie**
  - This class provides the functionality of using cookies.
  - It provides a lots of useful methods for cookies.

Constructor

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

Example

```
Cookie c= new Cookie("session_id","054");
```

Session Management: Cookies : Methods

void **setMaxAge**(int expiry)	Sets the maximum age in seconds for this Cookie
int **getMaxAge**()	Gets the maximum age in seconds of this Cookie. By default, -1 is returned, which indicates that the cookie will persist until browser shutdown.
String **getName**()	Returns the name of the cookie. The name cannot be changed after creation.
void **setValue**(String newValue)	Assigns a new value to this Cookie.
String **getValue**()	Gets the current value of this Cookie.
void **addCookie**(Cookie cookie)	Method of HttpServletResponse interface is used to add cookie in response object.
Cookie[] **getCookies()**	Returns an array containing all of the Cookie objects the client sent with this request. This method returns null if no cookies were sent.

Session Management: Cookies

- How to create Cookie?

Example

```
//creating cookie object
Cookie c= new Cookie("session_id","054");
//adding cookie in the response
response.addCookie(c);
```

Example

```
Cookie c[]=request.getCookies();
for(int i=0;i<c.length;i++)
{
out.print(c[i].getName()+””+c[i].getValue());
       //printing name&value of cookie
}
```

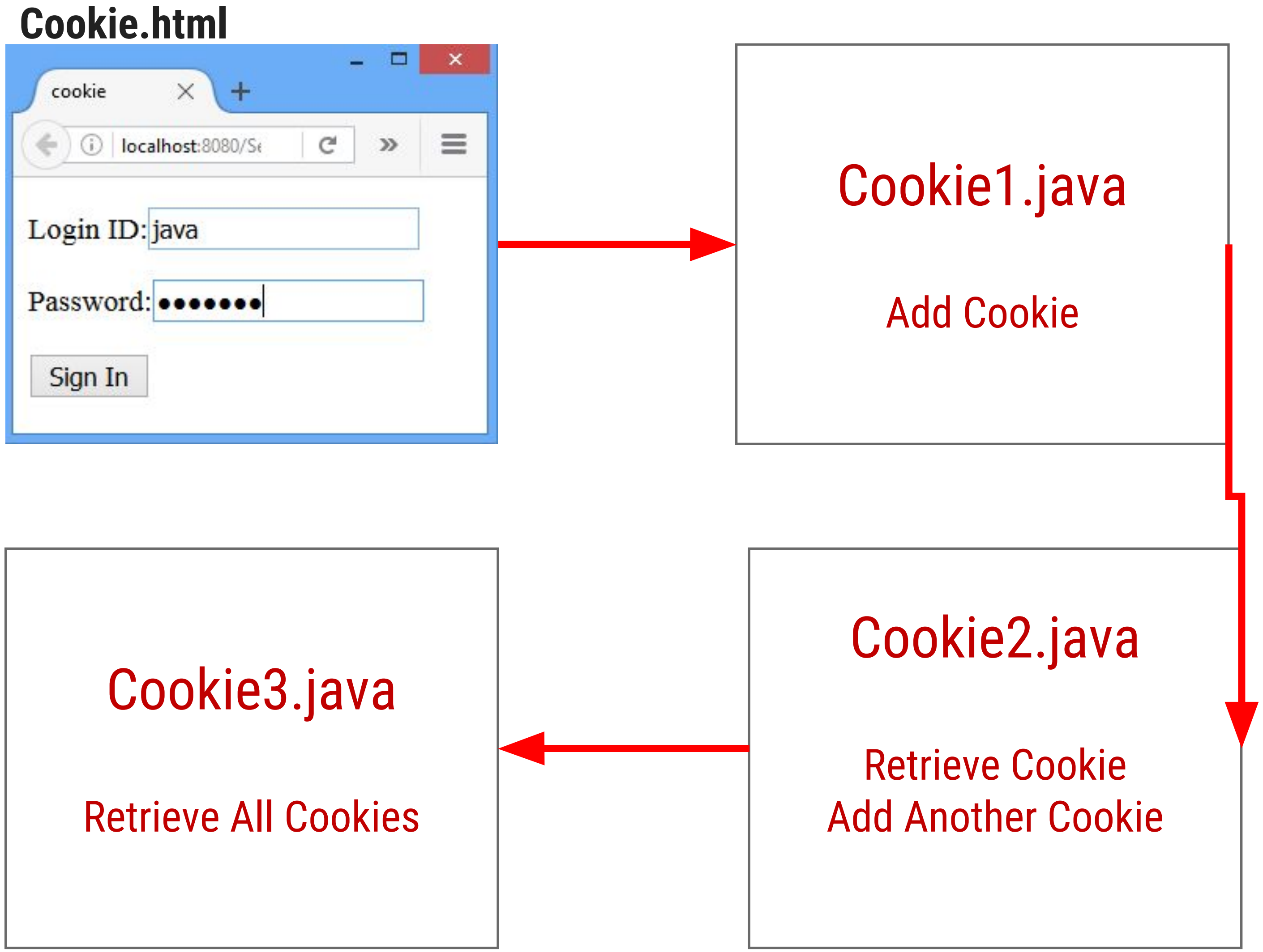
Session Management: Cookies

- **How to delete Cookie?**
    - Read an already existing cookie and store it in Cookie object.
    - Set cookie age as zero using **setMaxAge()** method to delete an existing cookie
    - Add this cookie back into response header.

Example

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

Session Management: Cookies



Cookie.html

Login ID: java

Password: •••••••

Sign In

Cookie1.java

Add Cookie

Cookie2.java

Retrieve Cookie
Add Another Cookie

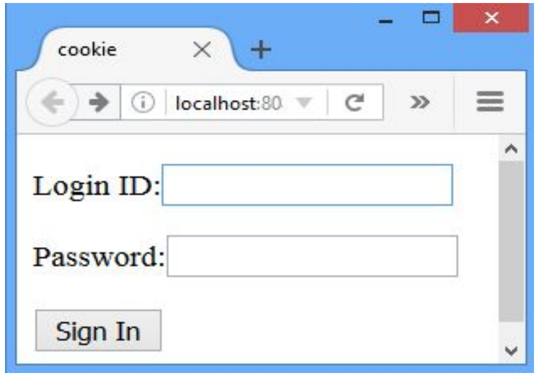
Cookie3.java

Retrieve All Cookies

Session Management: Cookies

cookie.html

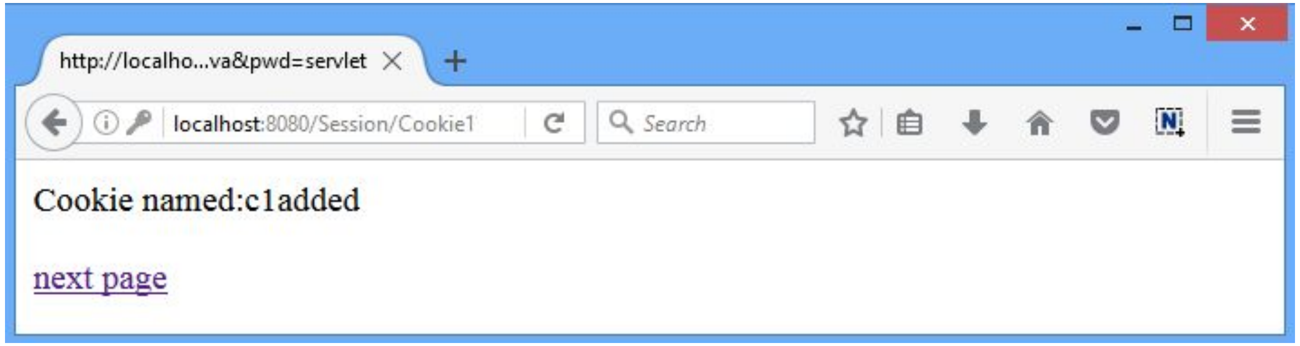
```
<html>
    <head>
        <title>cookie</title>
    </head>
    <body>
      <form action="/Session/Cookie1" >
    <p>Login ID:<input type="text" name="login"></p>
         <p>Password:<input type="password" name="pwd"></p>
         <p><input type="submit" value="Sign In"></p>
       </form>
    </body>
</html>
```



Session Management: Cookies

cookie1.java

```
public class Cookie1 extends HttpServlet
{   public void doGet(HttpServletRequest request, HttpServletResponse response)
                                              throws ServletException,IOException
    {   response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        String login=request.getParameter("login");
        String pwd=request.getParameter("pwd");
        if(login.equals("java") && pwd.equals("servlet"))
        {
        Cookie c = new Cookie("c1",login);//create cookie
            response.addCookie(c);//adds cookie with response
            out.println("Cookie named:"+c.getName()+" added");
            String path="/Session/Cookie2";
            out.println("<p><a href="+path+">next page</a></p>");
        }
        else {   //Redirect page to cookie.html}
} }
```

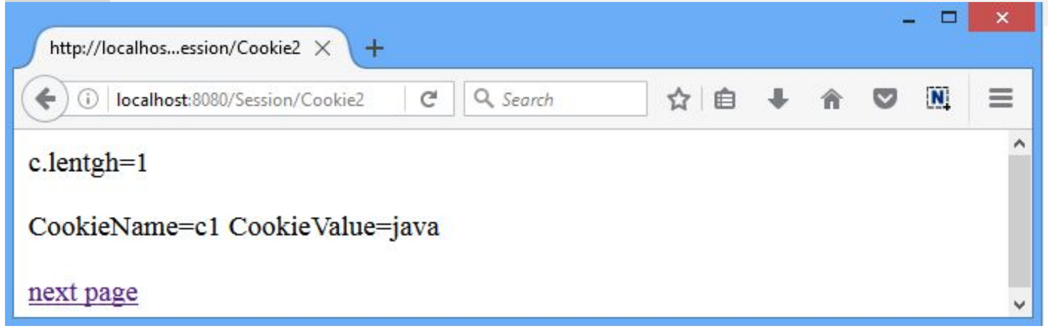

http://localho...va&pwd=servlet
localhost:8080/Session/Cookie1
Search

```
Cookie named:c1added

next page
```

Session Management: Cookies

cookie2.java

```
public class Cookie2 extends HttpServlet
{   public void doGet(HttpServletRequest request, HttpServletResponse response) throws
                          ServletException,IOException
    {   response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        Cookie c[]=request.getCookies();
        out.println("c.length="+c.length);
        for(int i=0;i<c.length;i++)
        {    out.println("CookieName="+c[i].getName()+
                      "CookieValue="+c[i].getValue());
    }
           //to add another cookie
        Cookie c1 = new Cookie("c2","054");
        response.addCookie(c1);
        String path="/Session/Cookie3";
        out.println("<a href="+path+">next page</a>");
    }
}
```

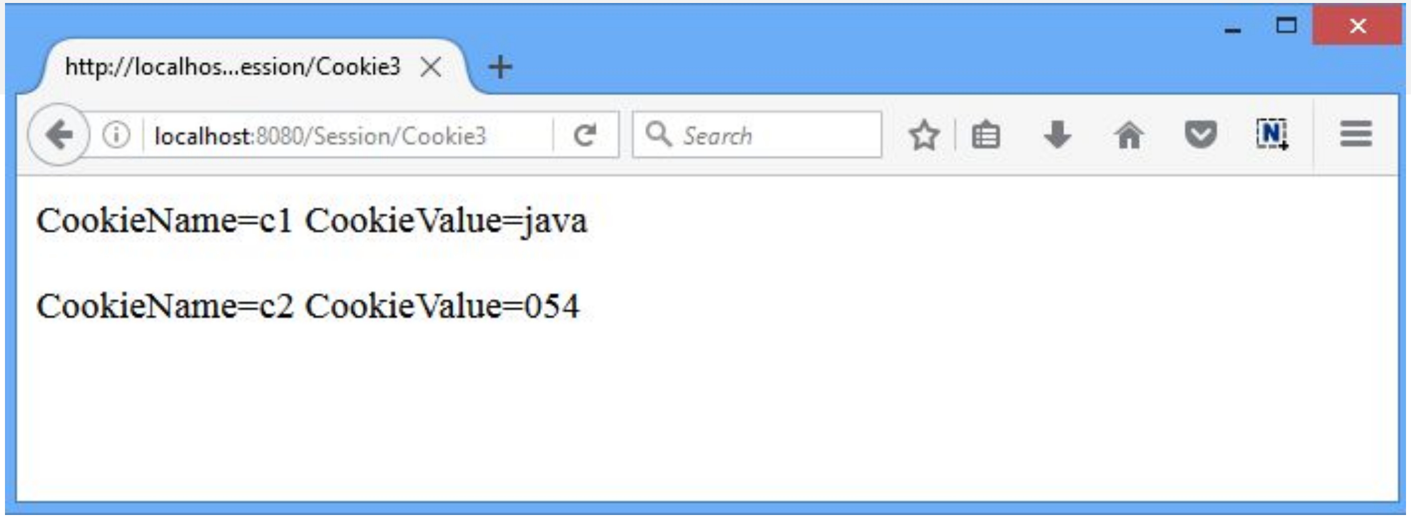

http://localhos...ession/Cookie2
localhost:8080/Session/Cookie2
Search

```
c.lentgh=1

CookieName=c1 CookieValue=java

next page
```

Session Management: Cookies

cookie3.java

```
public class Cookie3 extends HttpServlet
{   public void doGet(HttpServletRequest request, HttpServletResponse response)
                                    throws ServletException,IOException
    {   response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        Cookie c[]=request.getCookies();
        for(int i=0;i<c.length;i++)
        {   out.println("<p>");
            out.println("CookieName="+c[i].getName()+
                        "CookieValue="+c[i].getValue());
            out.println("</p>");
        }
    }
}
```


http://localhos...ession/Cookie3
localhost:8080/Session/Cookie3
Search

```
CookieName=c1 CookieValue=java

CookieName=c2 CookieValue=054
```

Session Management: Cookies

- **Advantage of Cookies**
  - Simplest technique of maintaining the state.
  - Cookies are maintained at client side.
- **Disadvantage of Cookies**
  - It will not work if cookie is disabled from the browser.
  - Only textual information can be set in Cookie object.

Session Management : HttpSession

- Apart from the above mentioned three ways, servlet provides HttpSession Interface which provides a way to identify a user across more than one page request

- The container creates a session id for each user.

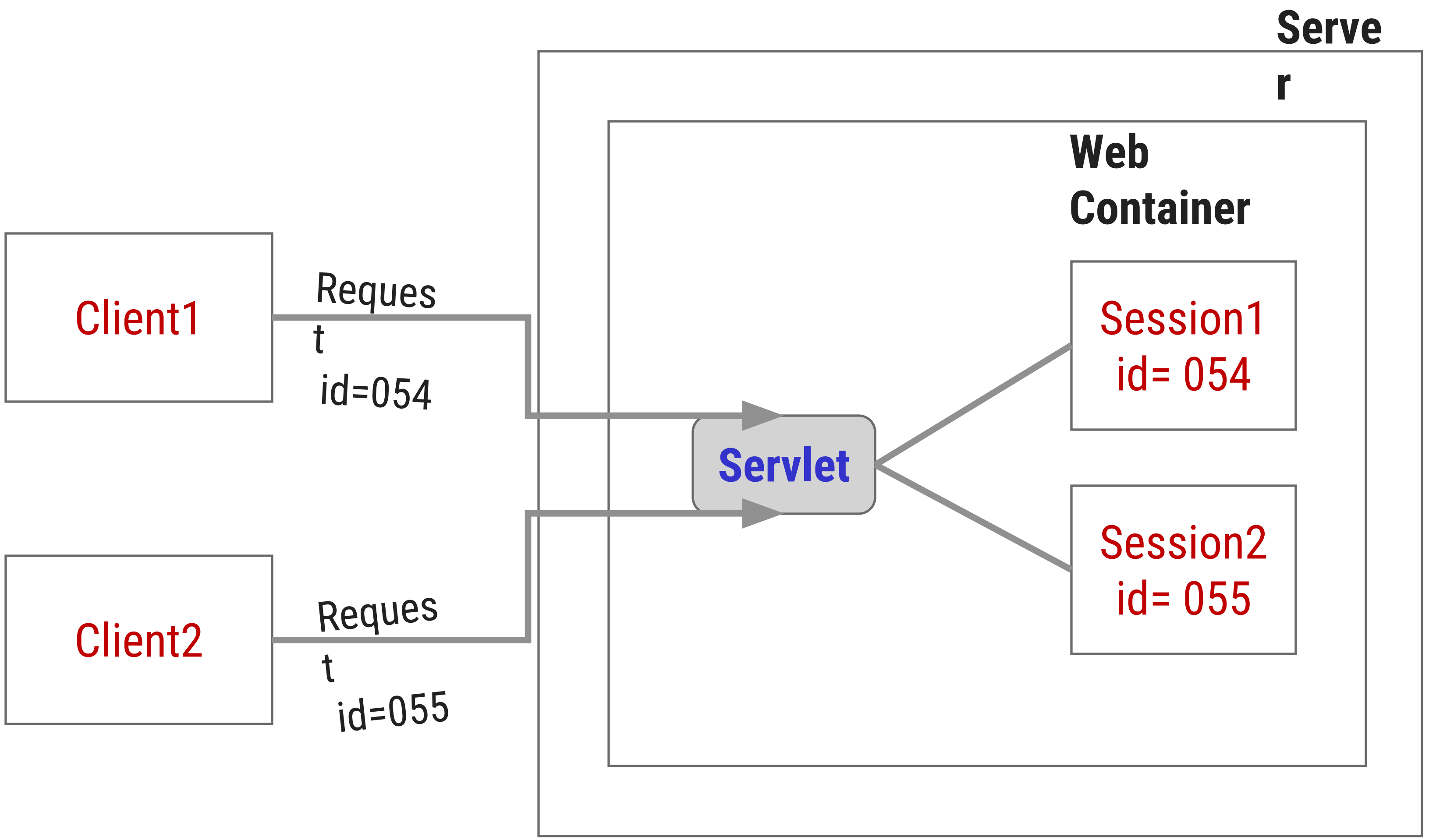
- The container uses this id to identify the particular user.

- An object of HttpSession can be used to perform two tasks:

    - Bind objects

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

Session Management : HttpSession



Working of HttpSession

Session Management :HttpSession

- Package: javax.servlet.http.**HttpSession**
- The servlet container uses this interface to create a session between an HTTP client and an HTTP server.
- In this technique create a session object at server side for each client.
- Session is available until the session time out, until the client log out.
- The default session time is 30 minutes and can configure explicit session time in web.xml file.
- The HttpServletRequest interface provides two methods to get the object of HttpSession

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

Session Management : HttpSession

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

Session Management : HttpSession

How to create the session?

```
HttpSession hs=request.getSession();
hs.setAttribute("s_id", "diet054");
```

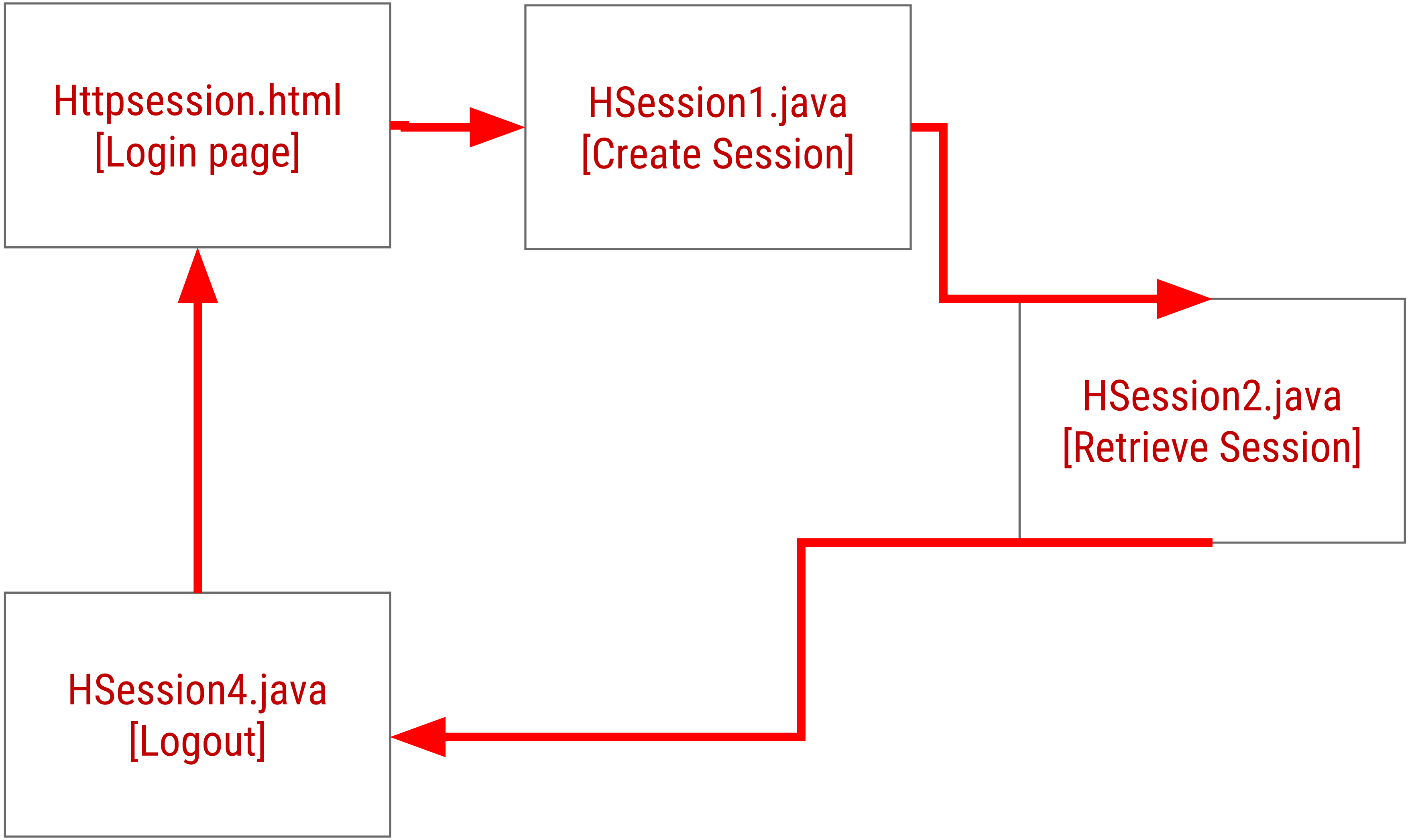
How to retrieve a session?

```
HttpSession hs=request.getSession(false);
String n=(String)hs.getAttribute("s_id");
```

How to invalidate a session?

```
hs.invalidate();
```

Session Management : HttpSession



Httpsession.html
[Login page]

HSession1.java
[Create Session]

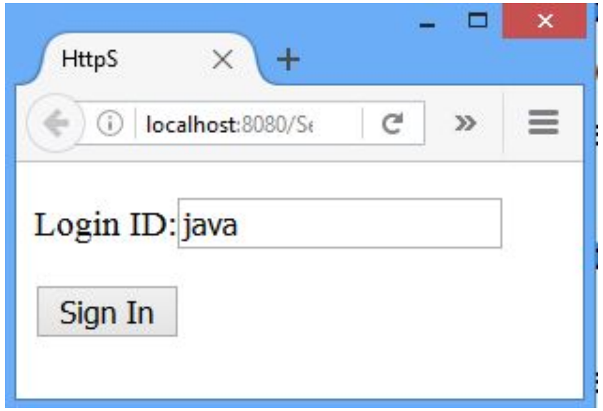
HSession2.java
[Retrieve Session]

HSession4.java
[Logout]

Session Management : HttpSession

Httpsession.html

```
<html>
    <head>
        <title>HttpSession</title>
    </head>
    <body>
       <form action="/Session/HSession1" method="Get">
            <p>Login ID:<input type="text" name="login"></p>
            <p><input type="submit" value="Sign In"></p>
        </form>
    </body>
</html>
```

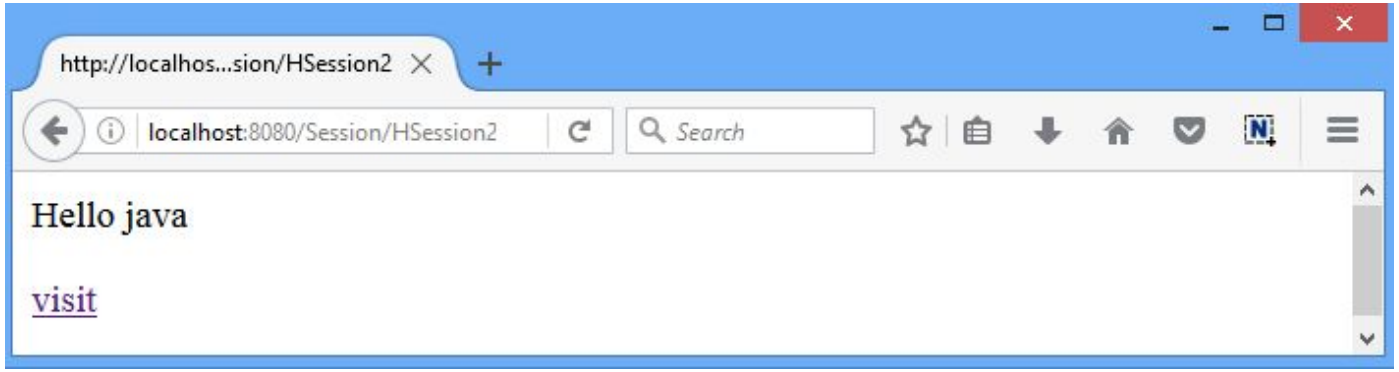


Login ID: java

Session Management : HttpSession

HSession1.java

```
response.setContentType("text/html");
PrintWriter out=response.getWriter();
RequestDispatcher rd;
String login=request.getParameter("login");
if(login.equals("java") )
{   HttpSession hs=request.getSession();
    hs.setAttribute("s_id",login);//set HttpSession
    out.println("Session Created");
    out.print("<a href='HSession2'>Homepage</a>");
}
else
{   out.println("<p><h1>Incorrect Login Id/Password
               </h1></p>");
    rd=request.getRequestDispatcher("/httpsession.html");
    rd.include(request, response);
}
```


http://localhos...sion/HSession2
localhost:8080/Session/HSession2
Search

Hello java

visit

Session Management : HttpSession

HSession2.java

```java
public class HSession2 extends HttpServlet
{   public void doGet(HttpServletRequest request, HttpServletResponse response)
                                    throws ServletException,IOException
   {
    response.setContentType("text/html");
    PrintWriter out=response.getWriter();
    HttpSession hs=request.getSession(false);
    String n=(String)hs.getAttribute("s_id");
    out.print("Hello "+n);
    out.print("<p><a hef='HSession3’>Logout</a></p>");
   }
 }

```

Session Management : HttpSession

HSession3.java

```
public class HSession3 extends HttpServlet
{   public void doGet(HttpServletRequest request, HttpServletResponse response)
                                   throws ServletException,IOException
   {
    response.setContentType("text/html");
    PrintWriter out=response.getWriter();
    HttpSession hs=request.getSession(false);
    hs.invalidate();// Session Invalidated
    try
    {
        String n=(String)hs.getAttribute("s_id");
    }
    catch(Exception ne)
        {
        out.println("Session Invalidated");
    }
    out.println("<form action='/Session/httpsession.html'>");
    out.println("<p><input type='submit’ value=‘Login'></p></form>");
   }
 }
```

Session Timeout

- The session timeout in a web application can be configured in two ways
  - Timeout in the deployment descriptor (web.xml)
  - Timeout with setMaxInactiveInterval()

Timeout in the deployment descriptor (web.xml)

```
<web-app>
    <session-config>
            <session-timeout> 10 </session-timeout>
    </session-config>
</web-app>
```

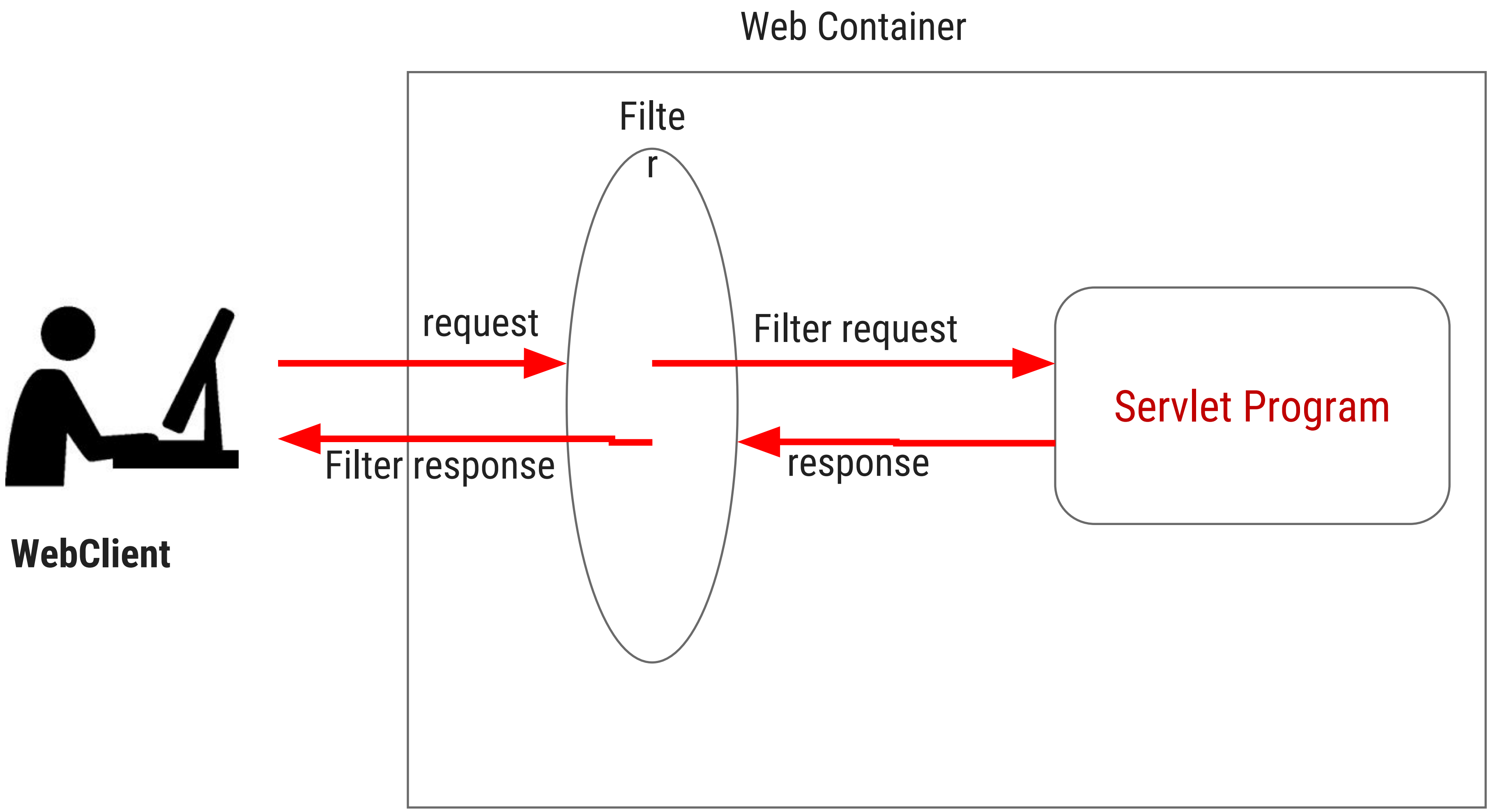
Here specified time is in **minutes**

Timeout with setMaxInactiveInterval()

```
HttpSession session = request.getSession();
session.setMaxInactiveInterval(10*60);
```

Here specified time is in **seconds**

Filter API


Web Container
Filte
r
request
Filter request
Servlet Program
Filter response
response
WebClient

Filter

- Filter is used for pre-processing of requests and post-processing of responses.
- Filters are configured in the deployment descriptor of a web application.
- Usage of Filter
    - Recording all incoming requests
    - Logs the IP addresses of the computers from which the requests originate
    - Conversion
    - Data compression
    - Encryption and Decryption
    - Input validation etc.

Filter API

- The javax.servlet package contains the three interfaces of Filter API.
  - Filter
  - FilterChain
  - FilterConfig

Filter Interface

- For creating any filter, you must implement the Filter interface.
- Filter interface provides the life cycle methods for a filter.
- Method

void **init**(FilterConfig config)	init() method is invoked only once. It is used to initialize the filter.
void **doFilter** (HttpServletRequest request, HttpServletResponse response, FilterChain chain)	doFilter() method is invoked every time when user request to any resource, to which the filter is mapped.It is used to perform filtering tasks.
void **destroy**()	This is invoked only once when filter is taken out of the service.

Filter Interface

Methods

```
public void init(FilterConfig config) throws ServletException {…}

public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)

            throws IOException,ServletException
{
    //filter logic…
}

public void destroy() {…}
```

FilterChain interface

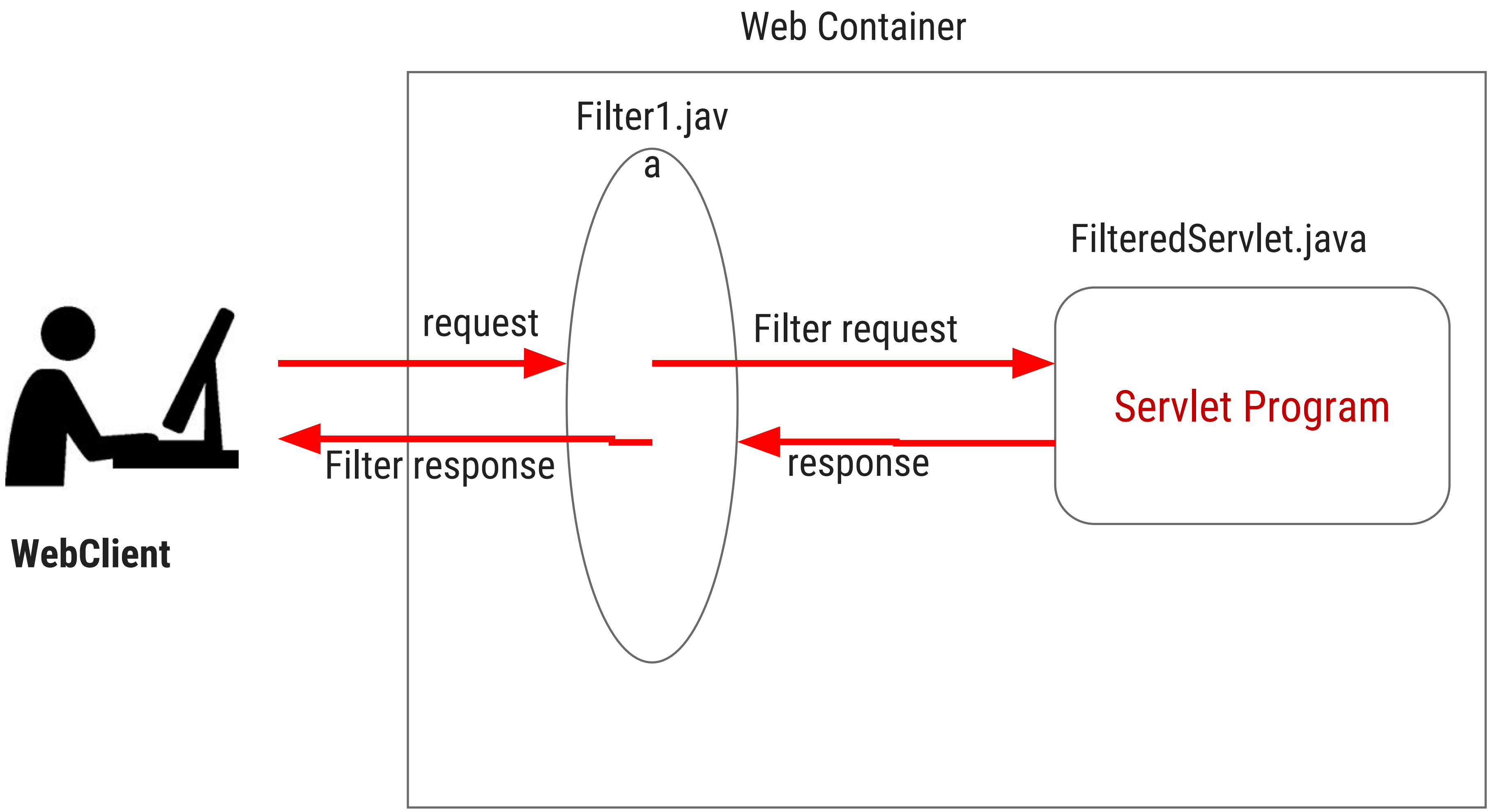
- The object of FilterChain is responsible to invoke the next filter or resource in the chain.
- This object is passed in the doFilter method of Filter interface.
- The FilterChain interface contains only one method:

void **doFilter** (HttpServletRequest request, HttpServletResponse response)	It passes the control to the next filter or resource.

Example

```
FilterChain chain;
chain.doFilter(req, resp);//send request to next resource
```

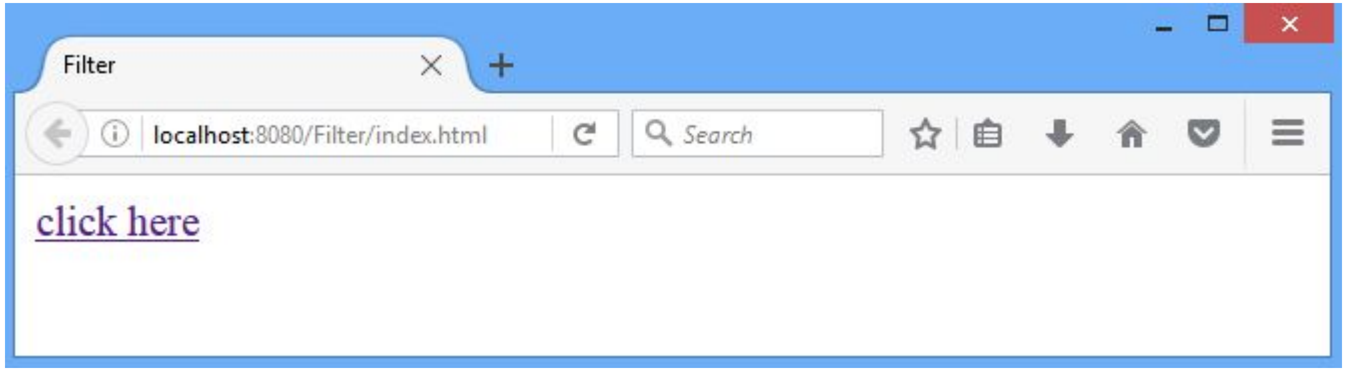
Filter Example



Filter Example: index.html

index.html

```
<html>
    <head>
        <title>Filter</title>
</head>
    <body>
        <a href="FilteredServlet">click here</a>
    </body>
</html>

```



Filter Example

Web.xml

```xml
<web-app>
<servlet>
   <servlet-name>FilteredServlet</servlet-name>
   <servlet-class>FilteredServlet</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>FilteredServlet</servlet-name>
   <url-pattern>/FilteredServlet</url-pattern>
</servlet-mapping>

<filter>
        <filter-name>f1</filter-name>
        <filter-class>Filter1</filter-class>
</filter>
<filter-mapping>
        <filter-name>f1</filter-name>
        <url-pattern>/FilteredServlet</url-pattern>
</filter-mapping>
```

Filter Example: Filter1.java

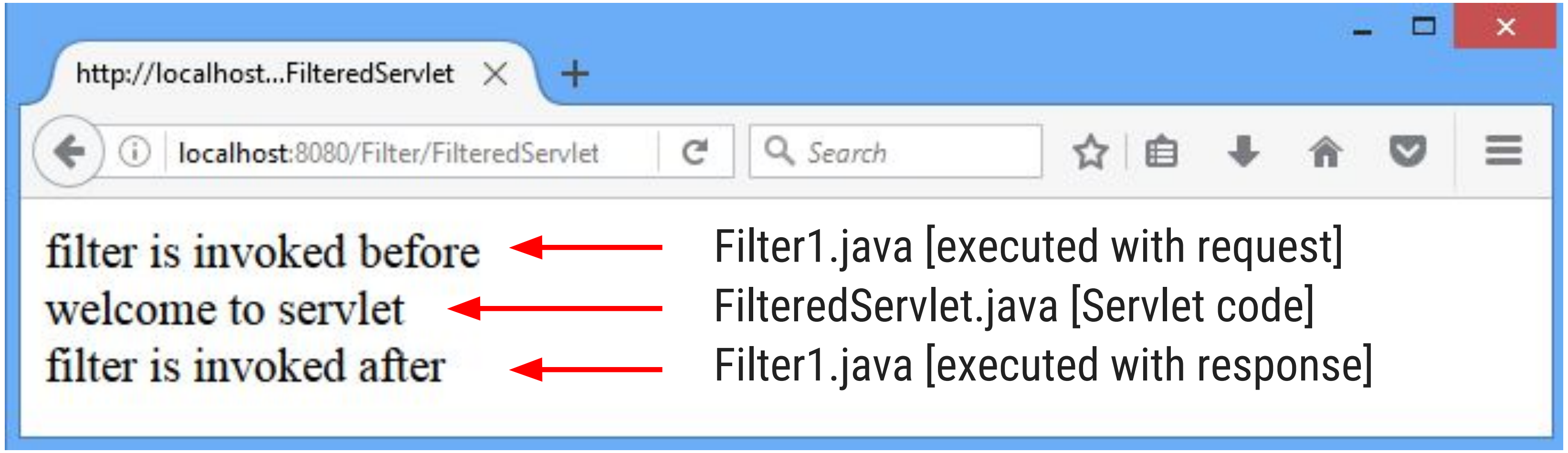
Filter1.java

```java
public class Filter1 implements Filter
{
public void init(FilterConfig arg0) throws ServletException {//overridden init() method}

public void doFilter(ServletRequest req, ServletResponse resp,FilterChain chain)
                                  throws IOException, ServletException
{
  PrintWriter out=resp.getWriter();
  out.print("filter is invoked before");//exe. with request
  chain.doFilter(req, resp);//send request to nextresource
  out.print("filter is invoked after");//exe. with response
}
public void destroy() {//overridden destroy() method}
}
```

Filter Example: FilteredServlet.java

FilterServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;
import javax.servlet.http.*;
public class FilteredServlet extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse response)
                              throws ServletException, IOException
   {
           response.setContentType("text/html");
           PrintWriter out = response.getWriter();
           out.println("<br>welcome to servlet<br>");
     }
}
```


http://localhost...FilteredServlet
localhost:8080/Filter/FilteredServlet
Search

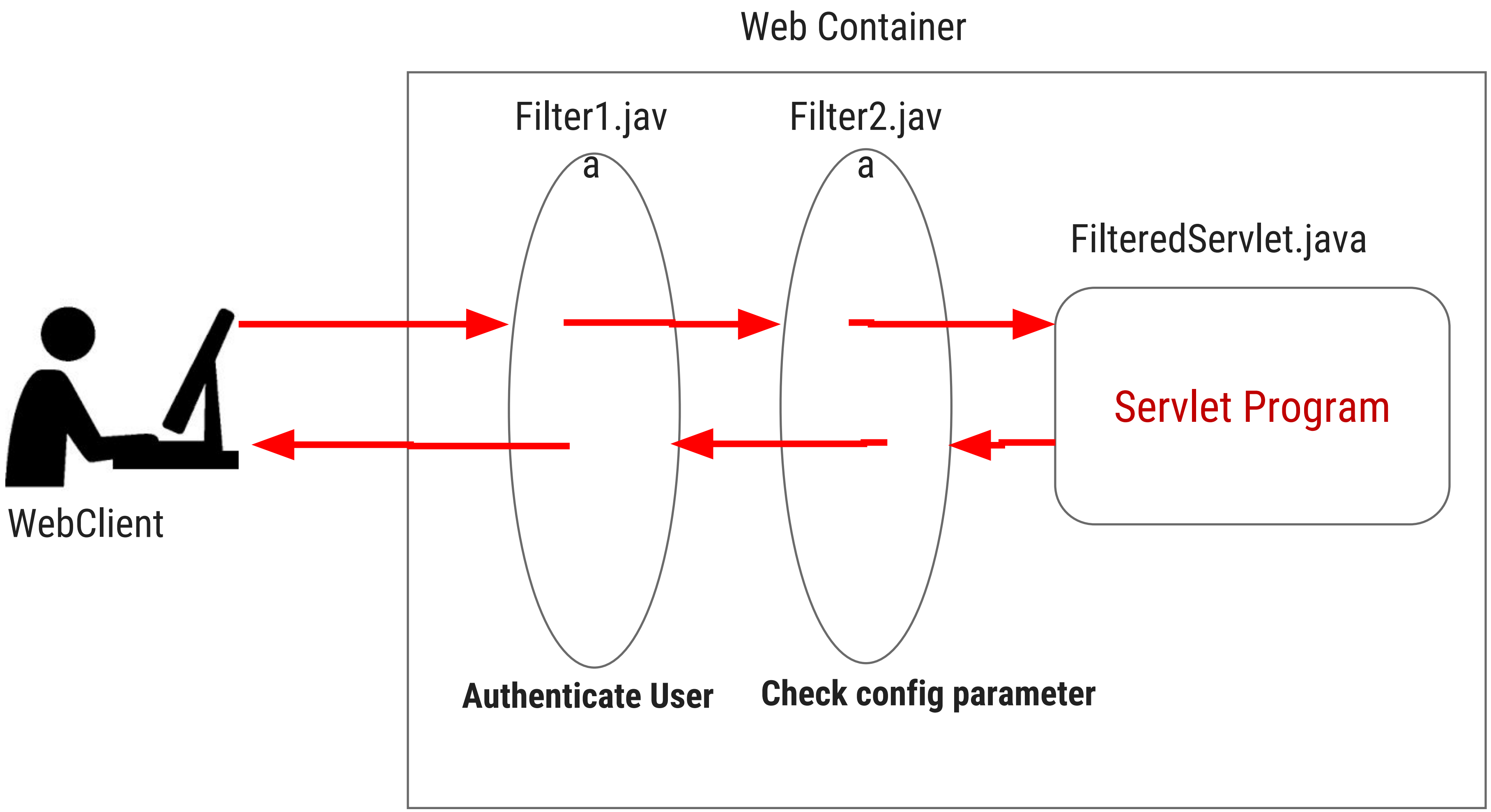
```
filter is invoked before
welcome to servlet
filter is invoked after
```

Filter1.java [executed with request]

FilteredServlet.java [Servlet code]

Filter1.java [executed with response]

Filter Example-2



Filter Example-2

index.html

```
<html>
    <head>
        <title>filter</title>
    </head>
 <body>
 <form action="/Filter/FilteredServlet" >
 <p>Login ID:<input type="text"   name="login"></p>
 <p>Password:<input type="password" name="pwd"></p>
 <p><input type="submit" value="Sign In"></p>
 </form>
 </body>
</html>
```

Filter Example-2

Web.xml

```
<web-app>
<servlet>
   <servlet-name>FilteredServlet</servlet-name>
   <servlet-class>FilteredServlet</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>FilteredServlet</servlet-name>
   <url-pattern>/FilteredServlet</url-pattern>
</servlet-mapping>

<filter>
        <filter-name>f1</filter-name>
        <filter-class>Filter1</filter-class>
</filter>
<filter-mapping>
        <filter-name>f1</filter-name>
        <url-pattern>/FilteredServlet</url-pattern>
</filter-mapping>
```

Web.xml

```
<filter>
        <filter-name>f2</filter-name>
        <filter-class>Filter2</filter-class>
        <init-param>
            <param-name>permit</param-name>
            <param-value>yes</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>f2</filter-name>
        <url-pattern>/FilteredServlet</url-pattern>
    </filter-mapping>
</web-app>
```

Filter Example-2

Filter1.java

```java
public class Filter1 implements Filter{

public void init(FilterConfig config) {}

public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)
                                        throws IOException, ServletException
{
  PrintWriter out=resp.getWriter();
  out.print("<p>filter1 is invoked before</p>");
  if(req.getParameter("login").equals("java") && req.getParameter("pwd").equals("servlet"))
  {
    chain.doFilter(req, resp);//send request to next resource
  }//if
  else
  {
    out.print("<p>invalid login/password</p>");
  }//else
  out.print("<p>filter1 is invoked after</p>");
}
public void destroy() {}
}
```

Filter Example-2

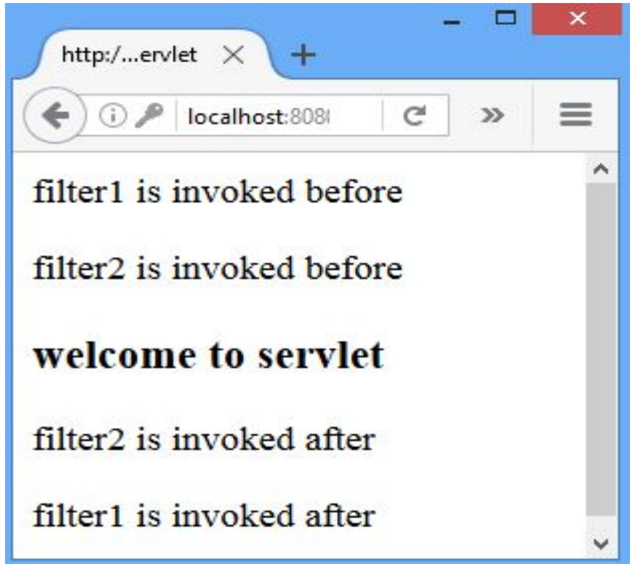
Filter1.java

```
public class Filter2 implements Filter{

String permission;
public void init(FilterConfig config) throws ServletException
{        permission=config.getInitParameter("permit");
}

public void doFilter(ServletRequest req, ServletResponse resp,FilterChain chain) throws IOException,
                                           ServletException
{        PrintWriter out=resp.getWriter();
         out.print("<p>filter2 is invoked before</p>");
              if(permission.equals("yes"))
              {      chain.doFilter(req, resp);}//if
              else
              {
             out.println("Permission Denied");
          }//else
         out.print("<p>filter2 is invoked after</p>");
}
public void destroy() {}}
```

Filter Example-2

FilteredServlet.java

```java
public class FilteredServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
                              throws ServletException, IOException
    {
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   out.println("<p><h3>welcome to servlet</h3></p>");
    }
}

```



filter1 is invoked before

filter2 is invoked before

welcome to servlet

filter2 is invoked after

filter1 is invoked after

Filter

- **Advantage of Filter**
  - Filter is pluggable.
  - One filter don't have dependency onto another resource.
  - Less Maintenance Cost
  - The **servlet filter is pluggable**, i.e. its entry is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet.
  - So maintenance cost will be less.

Servlet with JDBC

ServletWithJDBC.java

```java
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class JDBCServlet extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse response)
                                             throws ServletException,IOException
    {   response.setContentType("text/html");
        PrintWriter out=response.getWriter();
          //Program continued in next slide  ...
     try{
     Class.forName("com.mysql.jdbc.Driver");
     Connection con=DriverManager.getConnection ("jdbc:mysql://localhost:3306/ajava","root","");
     Statement st=con.createStatement();
     ResultSet rs=st.executeQuery("select * from cxcy");
     while(rs.next())
     {    out.println("<p>"+rs.getInt(1));
               out.println(rs.getString(2));
               out.println(rs.getString(3)+"</p>");
     }
     }catch(Exception e)
     {out.println("<p>inside exception"+e.toString()+"</p>");}
     }//doGet()
}//Class
```