Servlet API

There are 2 packages in Servlet API

1. **javax.servlet**
2. **javax.servlet.http**

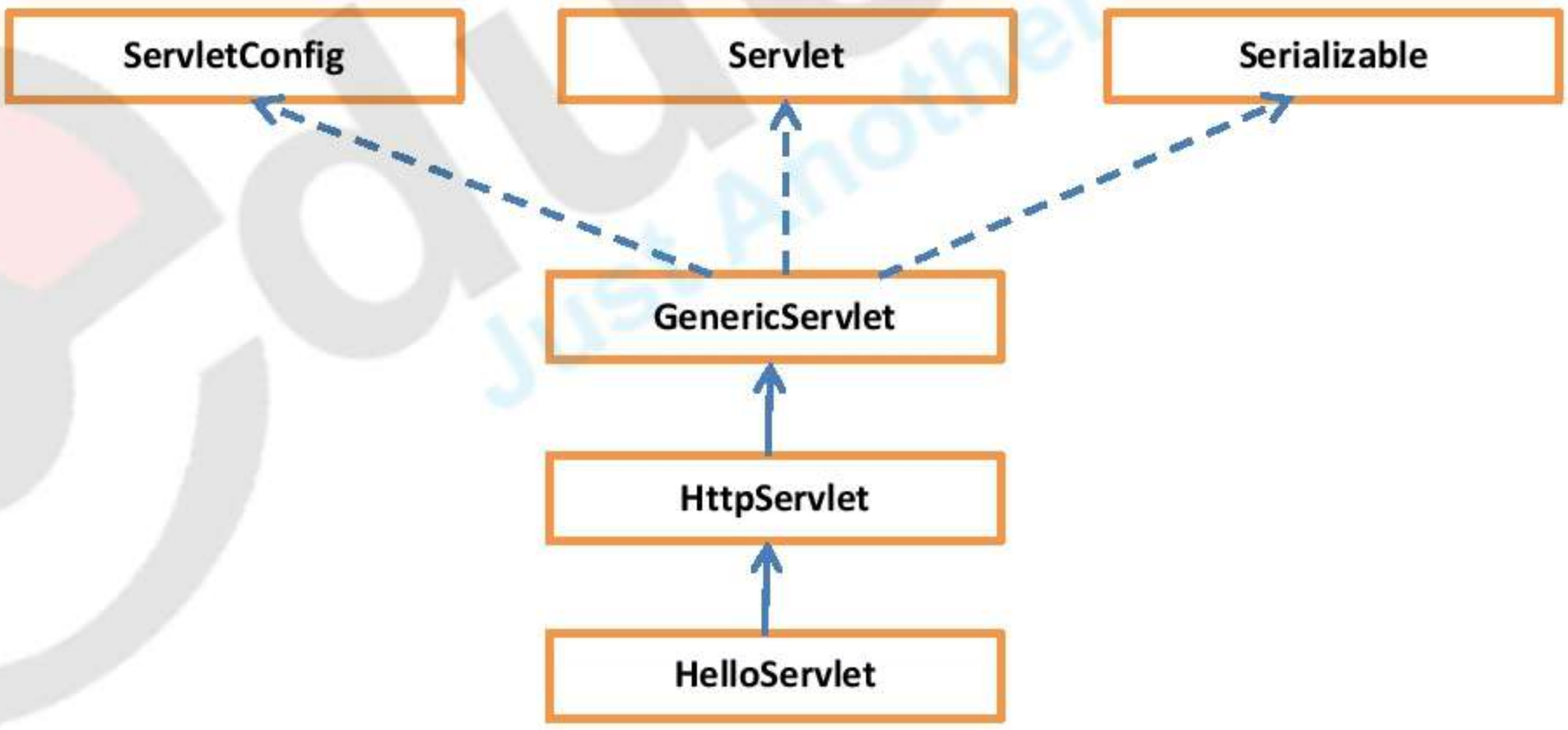
1. javax.servlet

- It contains many interfaces and classes that are used by the servlet or web container.
- **These are not specific to any protocol.**

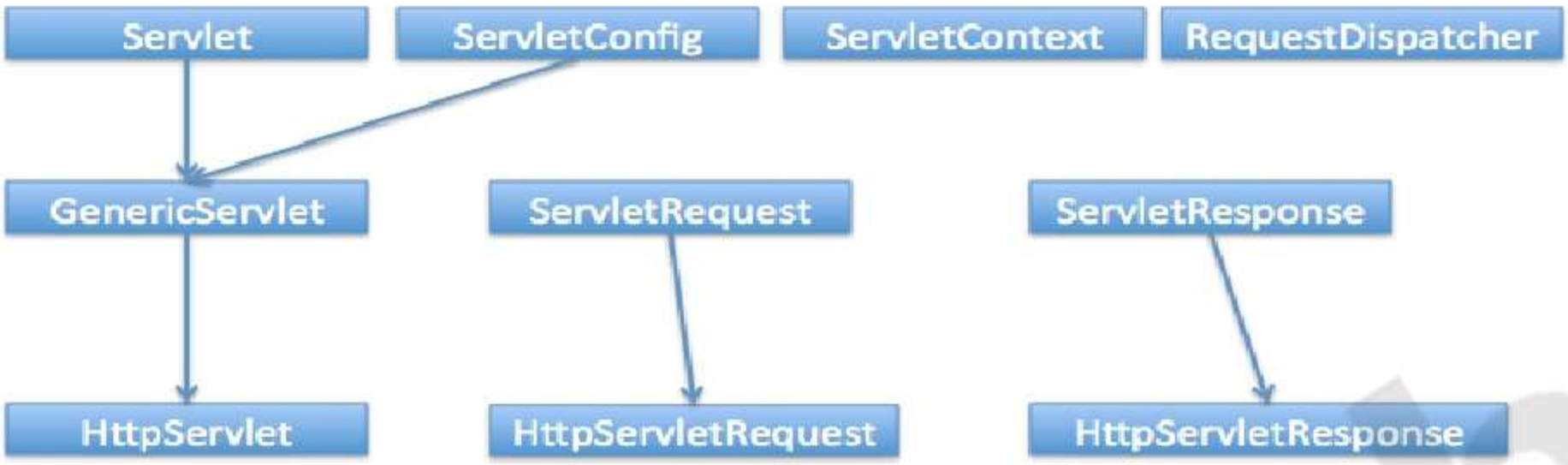
2. javax.servlet.http

- It contains interfaces and classes that are responsible for http requests only.

Architecture of a Servlet



Servlet API hierarchy



1. Serializable :

Serializable is a marker interface which doesn't have any methods. It is available in java.io package.

2. Servlet: (javax.servlet.Servlet)

- A servlet is a small Java program that runs within a Web server. Servlets receive and respond to requests from Web clients, usually across HTTP, the HyperText Transfer Protocol.

- Servlet is the base interface of **Java Servlet API**. Servlet interface declares the life cycle methods of servlet. All the servlet classes are required to implement this interface.
- This interface defines methods to initialize a servlet, to service requests, and to remove a servlet from the server. These are known as life-cycle methods and are called in the following sequence:

1. The servlet is constructed (Servlet object is created), and then initialized with the **init** method.
2. Any calls from clients to the **service** method are handled.
3. The servlet is taken out of service, then destroyed with the **destroy** method, then garbage collected and finalized.

Servlet Interface contains 5 methods.

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

Method	Description
public void init (ServletConfig config)	Called by the servlet container to indicate to a servlet that the servlet is being placed into service. It initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
public void service (ServletRequest request, ServletResponse response)	Called by the servlet container to allow the servlet to respond to a request. It provides response for the incoming request. It is invoked at each request by the web container.
public void destroy()	Called by the servlet container to indicate to a servlet that the servlet is being taken out of service. It is invoked only once and indicates that servlet is being destroyed.
public ServletConfig getServletConfig()	returns the object of ServletConfig, which the servlet can use to get any startup information
public String getServletInfo()	which allows the servlet to return basic information about itself, such as author, version, and copyright

3. **ServletConfig**(javax.servlet.ServletConfig)

- ServletConfig is used to pass configuration information to Servlet.
- Every servlet has it's own ServletConfig object and servlet container is responsible for instantiating this object.
- ServletConfig is implemented by the servlet container **to initialize a single servlet** using init(). That is, you can pass initialization parameters to the servlet using the web.xml deployment descriptor.
- For understanding, this is similar to a constructor in a java class.

- We can provide servlet init parameters in **web.xml** file or through use of WebInitParamannotation. We can use **getServletConfig()** method to get the ServletConfig object of the servlet.

  This interface contains 4 methods

  1. getServletContext()
  2. getInitParameter()
  3. getInitParameterNames()
  4. getServletName()

4. GenericServlet (javax.servlet.GenericServlet)

- GenericServlet defines a generic, protocol-independent servlet.
- GenericServlet gives a blueprint and makes writing servlet easier.
- GenericServlet provides simple versions of the lifecycle methods init and destroy and of the methods in the ServletConfig interface.
- GenericServlet implements the log method, declared in the ServletContext interface.
- To write a generic servlet, it is sufficient to override the abstract service method.

- GenericServlet is getting 9 abstract methods from super interfaces.

- GenericServlet is abstract class which implements Servlet,ServletConfig and SerializableInterfaces.

- It provides the implementaion of all the methods of these interfaces except the service method.

- GenericServlet name itself says that it is Generic to all the protocols and it can handle any type of request so it is protocol-independent.

- Whatever methods are implemented in GenericServlet, those are protocol independent. All these methods' behavior is not varying from one protocol to another protocol.

- But service() method behavior(implementation) is varying from one protocol to another protocol. That is why it is declared as abstract method. Becoz of abstract service() method, GenericServlet is declared as abstract class.

5. **HttpServlet** (javax.servlet.http.HttpServlet)

- HttpServlet defines a HTTP protocol specific servlet.
- HttpServlet gives a blueprint for Http servlet and makes writing them easier.

- HttpServlet is exclusively for addressing HTTP requests. It is specific to only one protocol (i.e. HTTP Protocol).

- Inside HttpServlet, service() method is implemented according to HTTP Protocol.

- In HttpServlet, several new methods are added which are specific to HTTP Protocol.

- In HttpServlet, doGet() & doPost() methods are newly added

- Dont override service(). Try to override doGet() & doPost().

- Inside body of service() method,

  We could make (submit) HttpRequest in 2 ways.
  a. method = get
  b. method = post
  - In service() method, server is checking HttpRequest type.
  - If the method type is get, then server is calling doGet() method.
  - If the method type is post, then server is calling doPost() method.
  - If we don't mention method type, then by default the method type is 'get'.

- In doGet() & doPost() methods only one statement is there, (i.e.) throw new methodNotSupportedException.

- Overriding service() method is not at all advisable. Server is always calling service() method by default.
  While executing service() method, Server is checking method type,

  - If the method type is get, then server is calling doGet() method.
  - If the method type is post, then server is calling doPost() method.

- In our Servlet program, If doGet() method not overrided, then calling super class (HttpServlet) doGet() method. Inside that method 'methodNotSupportedException' is there, which is raising Exception.

- In our Servlet program, If doPost() method not overrided, then calling super class (HttpServlet) doPost() method. Inside that method 'methodNotSupportedException' is there, which is raising Exception.

6. ServletContext interface (javax.servlet.ServletContext)

- ServletContext interface provides access to web application variables to the servlet.
- ServletContext is implemented by the servlet container **for all servlet** to communicate with its servlet container
- The ServletContext is unique object and available to all the servlets in the web application.
- When we want some init parameters to be available to multiple or all of the servlets in the web application, we can use ServletContext object and define parameters in web.xml using**<context-param>** element.
- We can get the ServletContext object via the getServletContext() method of ServletConfig.
- Servlet engines may also provide context objects that are unique to a group of servlets and which is tied to a specific portion of the URL path namespace of the host.

7. ServletRequest interface

- ServletRequest interface is used to provide client request information to the servlet.
- Servlet container creates ServletRequest object from client request and pass it to the servlet service() method for processing.

8. ServletResponse interface

- ServletResponse is used by servlet in sending response to the client.
- Servlet container creates the ServletResponse object and pass it to servlet service() method and later use the response object to generate the HTML response for client.

9. RequestDispatcher interface

- RequestDispatcher is used to forward the request to another resource that can be HTML, JSP or another servlet in the same context.
- We can also use this to include the content of another resource to the response.
- This interface is used for servlet communication within the same context.

There are two methods defined in this interface:

1. **void forward(ServletRequest request, ServletResponse response)** - forwards the request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
2. **void include(ServletRequest request, ServletResponse response)** - includes the content of a resource (servlet, JSP page, HTML file) in the response.

- We can get RequestDispatcher in a servlet using ServletContext *getRequestDispatcher(String path)*method.
- The path must begin with a / and is interpreted as relative to the current context root.

Parameters and forms

Web programs broke with the classical idea of linked hypertext documents.The need arises to send data from Web pages to the server (eg username and password). When this happened both HTTP and HTML were modified to support sending parameters and forms from client to server. For this paper two methods called Get and Post were invented.

1.Get Method

The servlet receives a request and response object. The request object contains information about the HTTP request, plus parameters, and other header info. The response object lets you set response headers, cookies, and let you write to the output stream.

This technique consists of adding to the URL a sign "?" and pair of "**name**" and "**value**" separated by a symbol "**&**" .

Example:

http://localhost:7070/app1/servlet10?username=Sashi&password=Kumar

URL — Parameters (Query string)

The Get method has two main drawbacks.

1. The first is that the data are in the URL.
   This is very bad in the example username and password as anyone who saw the box of the URL in the browser see your key.

2. The second disadvantage is that the length of the URL is limited so this limits us the number of characters that can be sent.

Example for doGet()

(Scenerio 1)

test1.html

```
<form action = 'servlet10' name = 'SimpleForm' method = 'get'>
     <table>
          <tr>
               <td> Name      :          </td>
               <td> <input type = 'text' name ='username'
                                        value = 'Sashi' /></td>
          </tr>

          <tr>
               <td> Password  :    </td>
               <td> <input type = 'text' name = 'password'
                                        value = 'Kumar' /></td>
          </tr>
     </table>
          <input type = 'submit' value = 'sumbit' />
</form>
```

We can see that the label form has an attribute action that tells the server which page request will be sent. The attribute method indicates the Get method we discussed earlier to submit.

Labels input boxes represent text and the attribute name indicates the name of the parameter used to send the details of the server box.

Servlet10.java

```
public class Servlet10 extends HttpServlet
{
     public void doGet(HttpServletRequest req,
                       HttpServletResponse   res)
                  throws ServletException, IOException
     {
          PrintWriter out = res.getWriter();

          out.println("This is doGet() method");
          out.println("User Name : " + req.getParameter("username"));
          out.println("Password : " + req.getParameter("password"));
     }
}
```

Output:

Url :
http://localhost:7070/app1/servlet10?username=Sashi&password=Kumar

Here, username and password are displayed in URL in the browser

This is doGet() method

User Name : Sashi
Password : Kumar

Here,

<form action = 'servlet10' name = 'SimpleForm' method = ‘get’>

So doGet() method is called in Servlet10. doGet() is available in Servlet10. So it produces the above output

(Scenerio 2)

Suppose if,

<form action = 'servlet10' name = 'SimpleForm' method = ‘post’>

So doPost()method is called in Servlet10, doPost() is not overrided in Servlet10. So it produces the ‘405 Error’ (HTTP Status 405 - HTTP method POST is not supported by this URL)

Becoz inside doPost(), only one statement is there, (i.e.) new throws methodNotFoundException. So it raises Exception

2.Post Method

This method like the Get sent to the server pairs
of "**name**" and "**value**" but it does so within the HTTP message
requesting a page to the server.

Thus the data are not visible in the URL and there is no limitation given by the maximum length of a URL.The Post method is associated with the HTML form.

Example for doPost()

(Scenerio 1)
test2.html

```
<form action = 'servlet11' name = 'SimpleForm' method = 'post'>
     Name      :
     <input type = 'text' name ='username' value = 'Sashi' /></br>
     Password  :
     <input type = 'text' name = 'password' value = 'Kumar' /></br>
          <input type = 'submit' value = 'sumbit' />
</form>
```

Servlet11.java

```
public class Servlet11 extends HttpServlet
{
     public void doPost(HttpServletRequest req,
                        HttpServletResponse res)
                  throws ServletException, IOException
     {
          PrintWriter out = res.getWriter();

          out.println("This is doPost() method");
          out.println("User Name : " + req.getParameter("username"));
          out.println("Password  : " + req.getParameter("password"));

          out.println("Query String : " + req.getQueryString());
          out.println("URL :"+ req.getRequestURL()+req.getQueryString());
     }
}
```

Output:

```
Url  :    http://localhost:7070/app1/servlet10

Here, username and password are not displayed in URL in the browser.
Becoz we used method = 'post'


This is doPost() method
User Name    :      Sashi
Password     :      Kumar

Query String :      username=Sashi&password=Kumar
URL          :
    http://localhost:7070/app1/servlet10username=Sashi&password=Kumar

Here,
<form action = 'servlet11' name = 'SimpleForm' method = 'post'>

    So doPost() method is called in Servlet10. doPost () is
    available in Servlet11. So it produces the above output
```

(Scenerio 2)

```
Suppose if,

<form action = 'servlet11' name = 'SimpleForm' method = 'get'>

    So doGet()method is called in Servlet11, doGet() is not
    available in Servlet11.      So it produces the  '405 Error'
    (HTTP Status 405 - HTTP method GET is not supported by this URL)

    Becoz inside doGet(), only one statement is there,
    (i.e.) new throws methodNotFoundException. So it raises
    Exception
```

getRequestURL() Method

- It is available in HttpServletRequest Interface
- Reconstructs the URL the client used to make the request. The returned URL contains a protocol, server name, port number, and server path, but it does not include query string parameters.
- Because this method returns a StringBuffer, not a string, you can modify the URL easily, for example, to append query parameters.
- This method is useful for creating redirect messages and for reporting errors.

Differences between Get and Post:

GET	POST
1) In case of Get request, only **limited amount of data** can be sent because data is sent in header.	In case of post request, **large amount of data** can be sent because data is sent in body.
2) Get request is **not secured** because data is exposed in URL bar.	Post request is **secured** because data is not exposed in URL bar.
3) Get request **can be bookmarked**	Post request **cannot be** bookmarked
4) Get request is **idempotent.** It means second request will be ignored until response of first request is delivered.	Post request is **non-idempotent**
5) Get request is **more efficient** and used more than Post	Post request is **less efficient** and used less than get.

Importance of Initilization of servletConfig object

- ServletConfig objct exist *one* per servlet program.
- An object of ServletConfig created by the container during its initialization phase.
- An object of ServletConfig is available to the servlet during its execution, once the servlet execution is completed, automatically ServletConfig interface object will be removed by the container.

Skeleton of GenericServlet abstract class

```
Public abstract class GenericServlet
        Implements Servlet, ServletConfig, java.io.Serializable
{
        Public ServletConfig config;

        Public void init(ServletConfig config)throws ServletException
        {
                this.config= config;
                init();
        }
        Public void init() {

        }

        Public abstract void service(ServletRequest request,
                                     ServletResponse response);

        Public void destroy() {
        }

        public ServletConfig getServletConfig() {
                return config;
        }

        Public ServletContext getServletContext () {
                Return getServletConfig().getServletContext();
        }
}
```

- Here GenericServlet is implementing all the abstract methods except service(). That's why it was declared as abstract class.

- We can found two init() methods { init(ServletConfig config) and init() } as part of GenericServlet.

- init(ServletConfig config) is the first init method and servlet container always calls this method. Here global attribute servletConfig object is getting initialized. Then it calls the second init method (init()) which has no argument.

- If we want to perform some task when ever servlet object got created , we need to write code as part of init(). Our servlet will override init(), but container will always call init(ServletConfig config) of GenericServlet, which will call init() of our servlet.

Why init() over init(ServletConfig config)?

It's easy and recommended to override init() than init(ServletConfig config) since there is no need to call super.init(config) if we use init().

Case 1 :

Servlet1.java

```
public class Servlet1 extends HttpServlet {

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

          PrintWriter out = response.getWriter();

          out.print("This is service() method");
     }
}
```

Here many methods are inherited from HttpServlet,
We'll see 2 methods,

```
Public void init(ServletConfig config) {
          this.config = config;
          init();
}
  Public void init(){

  }
```

Server is automatically calling init(servletConfig) method & passing servletConfig object to this method. **Here global attribute (Servletconfig object) is initialized.** From the body of init(servletConfig) method, init() method is called. But init() method has no statements.

After executing this init() method, Servler is calling service() method.

In this case Running is Success.

Output:
This is service() method

Case 2 :

```
public class Servlet1 extends HttpServlet {

      @Override
      Public void init() {
            System.out.println("This is init() method");
      }

      Protected void service(HttpServletRequest request,
                             HttpServletResponse response)
                  throws ServletException, IOException {

            PrintWriter out = response.getWriter();

            out.print("This is service() method");
      }
}
```

Here init(servletConfig), init() methods are inherited from HttpServlet. We're overriding init() method which has no arg.

Server is automatically calling init(servletConfig) method, From body of this method, overrided init() method is called, then Servler is calling service() method.

In this case Running is Success.

Output:
This is init() method
This is service() method

Case 3 :

```
public class Servlet1 extends HttpServlet {

      @Override
      Public void init(ServletConfig config) {

      }

      Protected void service(HttpServletRequest request,
                             HttpServletResponse response)
                        throws ServletException, IOException {

            PrintWriter out = response.getWriter();

            out.print("This is service() method ");
      }
```

Here init(servletConfig), init() methods are inherited from HttpServlet. We're overriding init(servletConfig) method.

Server is automatically calling overrided init(servletConfig) method. But inside init(ServletConfig config) method, global attribute (servletConfig object) not initialized.

If we call any inherited methods, we'll get NullPointerException. Because global attribute (servletConfig object) not initialized.

Global attribute(Servletconfig object)should initialized at anycost.

Output:

HTTP Status 500 - Servlet.init() for servlet Servlet1 threw exception java.lang.NullPointerException

Case 4 :

```
public class Servlet1 extends HttpServlet
{

      @Override
      Public void init(ServletConfig config) throws ServletException
      {

            super.init(config);
            System.out.print("This is init((ServletConfig config)");

            ServletContext context =  getServletConfig().getServletContext();
      }

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

            PrintWriter out = response.getWriter();

            System.out.println("This is doGet() method");
      }
}
```

Here init(servletConfig), init() methods are inherited from HttpServlet. We're overriding init(servletConfig) method.

Server is automatically calling overrided init(servletConfig) method. But inside init(ServletConfig config) method, We're calling super class(HttpServlet) init(ServletConfig config) method. In this super class method global attribute (servletConfig object) is initialized.

In this case Running is Success.

Output:

```
This is init((ServletConfig config)
This is doGet() method
```

Conclusion:

Don't override the 1st init(ServletConfig) method. This method is only for initializing global attribute (servletConfig object).

Always override the 2nd init() method which takes no argument.
Here Servlet container always calls the inherited init(ServletConfig) method, global attribute is got initialized. From body of this method, overrided init() method is called.

RequestDispatcher

- It is used dispatch one request to another resource which is available in same web application. That another resource can be Servlet, JSP, or HTML.
- RequestDispatcher dispatching current request from one Servlet to another Servlet in server without browser's intention.
- RequestDispatcher is a Servler side technique. Becoz browser does not aware of where request is going.
- RequestDispatcher is an interface in javax.servlet package.
- Whatever members are available in javax.servletthose are protocol independent.RequestDispatcher also available in javax.servlet. So this also protocol independent.
- **Within current Application only we can forward, or include request through RequestDispatcher.** We cannot use RequestDispatcher for other web applications.

There are 2 methods available in RequestDispatcher.

1. void **forward**(ServletRequest request, ServletResponse response)
2. void **include**(ServletRequest request, ServletResponse response)

We cannot create object to RequestDispatcher, Becoz it is an interface.

We can get RequestDispatcher Object in 2 ways.

1. Relative way by using request Object
2. Absolute way by using ServletContext Object

1.Relative way by using request Object

```
RequestDispatcher rd = request.getRequestDispatcher("ServletName");
```

If we use request object from Servlet1, request & response directly going into Servlet2. This is Relative way.

2.Absolute way by using ServletContext Object

```
ServletContext context = getServletContext();
RequestDispatcher rd = context.getRequestDispatcher("/ServletName ");
```

(or)

```
RequestDispatcher rd = getServletContext().context.getRequestDispatcher("/ServletName ");
```

If we use use Absolute way, From Servlet1 flow is not going to Servlet2 directly. It is going to root of the Application, & finding the Servlet2, then going into Servlet2.

To mention root, we're using (/). i.e. /Servlet2

Eg : 1

1.forward() method

- It forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server
- It forwards the request made by the client by the the resource (any of them servlet , jsp, html, etc.) on the server without including the content of the requested resource.
- Last Servlet output only displayed in the Browser.

test1.html:

```
<Form action="Servlet1">
        Parameter    :      <inputtype="text"name="param1"/>
                            <inputtype="submit"value="submit"/>
</ Form >
```

Servlet1.java:

```
public class Servlet1 extends HttpServlet {
      protected void doGet(HttpServletRequest request,
                           HttpServletResponse response)
                     throws ServletException, IOException {
            PrintWriter out = response.getWriter();
            String s1 = request.getParameter("param1");

            System.out.println("--------Console : Servlet1 Begin--------");
            out.println("--------Browser : Servlet1 Begin --------<br>");

            System.out.println("Console : Servlet1 : Parameter = " + s1);
            out.println("Browser : Servlet1 : Parameter = " + s1 + "<br>");

            String revParam1 = newStringBuffer(s1).reverse().toString();
            // Setting the revParam1 in response object
            request.setAttribute("result", revParam1);
            // We can follow any one of the way

            // 1. Using Relative way
            //RequestDispatcher rd = request.getRequestDispatcher("Servlet2");
            //Sending the request & response object to Servlet2
            //rd.forward(request, response);

            // 2. Using Absolute way
            ServletContext context = getServletContext();
            RequestDispatcher rd = context.getRequestDispatcher("/Servlet2");
            rd.forward(request, response);

            System.out.println("--------Console : Servlet1 End--------");
            out.println("--------Browser : Servlet1 End --------<br>");
      }
}
```

Servlet2.java:

```
public class Servlet2 extends HttpServlet {
      @Override
      Protected void doGet(HttpServletRequest request,
                           HttpServletResponse response)
                     throws ServletException, IOException {

            PrintWriter out = response.getWriter();

            System.out.println("--------Console : Servlet2 Begin--------");
            out.println("--------Browser : Servlet2 Begin --------<br>");

            //Reading Form data by using same request object
            String s2 = request.getParameter("param1");

            System.out.println("Console : Servlet2 : Parameter = " + s2);
            out.println("Browser : Servlet2 : Parameter = " + s2 + "<br>");

            // Reading the revParam from Servlet1 request object
            String revParam2 = (String) request.getAttribute("result");

            System.out.println("Console : Servlet2 : revParam = " + revParam2);
            out.println("Browser : Servlet1 : revParam = " + revParam2 + "<br>");

            System.out.println("--------Console : Servlet2 End--------");
            out.println("--------Browser : Servlet2 End --------<br>");
      }
}
```

Output :

url :http://localhost:7070/App2/test1.html

Parameter : Sashi Submit → Click

url :http://localhost:7070/App2/Servlet1?param1=Sashi

Browser Output:

--------Browser : Servlet2 Begin --------
Browser : Servlet2 : Parameter = Sashi
Browser : Servlet1 : revParam = ihsaS
--------Browser : Servlet2 End --------

Console Output:

```
--------Console : Servlet1 Begin--------
Console : Servlet1 : Parameter = Sashi
--------Console : Servlet2 Begin--------
Console : Servlet2 : Parameter = Sashi
Console : Servlet2 : revParam = ihsaS
--------Console : Servlet2 End--------
--------Console : Servlet1 End--------
```

Explanation:

In Servlet1 we're reading form data, (i.e) read '*param1*' value. (Sashi). Reversing the s1 value and storing in "revParam1". Adding that "revParam1"value into request object. Then defining RequestDispatcher object for "Servlet2", & calling forward() method by providing request & response objects.

Whenever we're calling forward() method, current request is dispatching to "Servlet2". Execution goes to "Servlet2" from "Servlet1".

In "Servlet2" we're reading the form data & printing. And getting "Servlet1" content (revParam1) from "request" object by using request.getAttribute() method.

Once "Servlet2" execution got over, then control goes back to "Servlet1". Remaining "Servlet1" Statements are executing.

Here last Servlet(Servlet2) output only displayed in the Browser.

Eg :2

test1.html:

```
<Form action="Servlet1">
        Parameter    :      <inputtype="text"name="param1"/>
                            <inputtype="submit"value="submit"/>
</ Form >
```

Servlet1.java:

```
public class Servlet1 extends HttpServlet {
      protected void doGet(HttpServletRequest request,
                           HttpServletResponse response)
                     throws ServletException, IOException {

            PrintWriter out = response.getWriter();
            String s1 = request.getParameter("param1");

            System.out.println("--------Console : Servlet1 Begin--------");
            out.println("--------Browser : Servlet1 Begin --------<br>");

            System.out.println("Console : Servlet1 : Parameter = " + s1);
            out.println("Browser : Servlet1 : Parameter = " + s1 + "<br>");

            String revParam1 = newStringBuffer(s1).reverse().toString();
            // Setting the revParam1 in response object
            request.setAttribute("result", revParam1);

            // By Using Relative way
            RequestDispatcher rd = request.getRequestDispatcher("Servlet2");
            //Sending the request & response object to Servlet2
            rd.forward(request, response);

            System.out.println("--------Console : Servlet1 End--------");
            out.println("--------Browser : Servlet1 End --------<br>");

      }
}
```

Servlet2.java:

```
public class Servlet2 extends HttpServlet {
	@Override
	Protected void doGet(HttpServletRequest request,
				HttpServletResponse response)
			throws ServletException, IOException {
		PrintWriter out = response.getWriter();

		System.out.println("--------Console : Servlet2 Begin--------");
		out.println("--------Browser : Servlet2 Begin --------<br>");

		//Reading Form data by using same request object
		String s2 = request.getParameter("param1");

		System.out.println("Console : Servlet2 : Parameter = " + s2);
		out.println("Browser : Servlet2 : Parameter = " + s2 + "<br>");

		// By using Absolute way.
		RequestDispatcher rd1 =
				getServletContext().getRequestDispatcher("/Servlet3");
		rd1.forward(request, response);

		System.out.println("--------Console : Servlet2 End--------");
		out.println("--------Browser : Servlet2 End --------<br>");
	}
}
```

Servlet3.java:

```
public class Servlet3 extends HttpServlet {
	protected void doGet(HttpServletRequest request,
				HttpServletResponse response)
			throws ServletException ,IOException {

		PrintWriter out = response.getWriter();

		System.out.println("--------Console : Servlet3 Begin--------");
		out.println("--------Browser : Servlet3 Begin --------<br>");

		//Reading Form data by using same request object
		String s3 = request.getParameter("param1");

		System.out.println("Console : Servlet3 : Parameter = " + s3);
		out.println("Browser : Servlet3 : Parameter = " + s3 + "<br>");

		// Reading the revParam from Servlet1 request object
		String revParam3 = (String) request.getAttribute("result");

		System.out.println("Console : Servlet3 : revParam3 = " + revParam3);
		out.println("Browser : Servlet3 : revParam3 = " + revParam3 + "<br>");

		System.out.println("--------Console : Servlet3 End--------");
		out.println("--------Browser : Servlet3 End --------<br>");
	}
}
```

Output :

url :http://localhost:7070/App2/test1.html

url :http://localhost:7070/App2/Servlet1?param1=BUDDY

Browser Output:

--------Browser : Servlet3 Begin --------
Browser : Servlet3 : Parameter = BUDDY
Browser : Servlet3 : revParam3 = YDDUB
--------Browser : Servlet3 End --------

Console Output:

```
--------Console : Servlet1 Begin--------
Console : Servlet1 : Parameter = BUDDY
--------Console : Servlet2 Begin--------
Console : Servlet2 : Parameter = BUDDY
--------Console : Servlet3 Begin--------
Console : Servlet3 : Parameter = BUDDY
Console : Servlet3 : revParam3 = YDDUB
--------Console : Servlet3 End--------
--------Console : Servlet2 End--------
--------Console : Servlet1 End--------
```

Explanation:

From HTML file control goes to Servlet1, From body of Servlet1 control goes to Servlet2, , From body of Servlet2 control goes to Servlet3.

Once Servlet3 got over, Servlet2 remaining portion is executing. Once Servlet2 got over, Servlet1 remaining portion is executing

Here last Servlet(Servlet2) output only displayed in the Browser.

2.include() method

- It includes the content of a resource (servlet, JSP page, HTML file) in the response
- Include() method includes all the Servlets output into the Browser.

Eg :3

test1.html:

```
<Form action="Servlet4">
      Parameter    :     <inputtype="text"name="param1"/>
                         <inputtype="submit"value="submit"/>
</ Form >
```

Servlet4.java:

```
public class Servlet4 extends HttpServlet {
      protected void doGet(HttpServletRequest request,
                           HttpServletResponse response)
                        throws ServletException, IOException {

            PrintWriter out = response.getWriter();
            String s1 = request.getParameter("param1");

            System.out.println("--------Console : Servlet4 Begin--------");
            out.println("--------Browser : Servlet4 Begin --------<br>");

            System.out.println("Console : Servlet4 : Parameter = " + s1);
            out.println("Browser : Servlet4 : Parameter = " + s1 + "<br>");

            String revParam1 = newStringBuffer(s1).reverse().toString();
            // Setting the revParam1 in response object
            request.setAttribute("result", revParam1);

            // By Using Absolute way
            ServletContext context = getServletContext();
            RequestDispatcher rd = context.getRequestDispatcher("/Servlet5");
            rd.include(request, response);

            System.out.println("--------Console : Servlet4 End--------");
            out.println("--------Browser : Servlet4 End --------<br>");

      }
}
```

Servlet5.java:

```
public class Servlet5 extends HttpServlet {
	@Override
	Protected void doGet(HttpServletRequest request,
				HttpServletResponse response)
			throws ServletException, IOException {

		PrintWriter out = response.getWriter();

		System.out.println("--------Console : Servlet5 Begin--------");
		out.println("--------Browser : Servlet5 Begin --------<br>");

		//Reading Form data by using same request object
		String s2 = request.getParameter("param1");

		System.out.println("Console : Servlet5 : Parameter = " + s2);
		out.println("Browser : Servlet5 : Parameter = " + s2 + "<br>");

		// By Using Relative way
		RequestDispatcher rd = request.getRequestDispatcher("Servlet6");
		//Sending the request & response object to Servlet2
		rd.include(request, response);

		System.out.println("--------Console : Servlet5 End--------");
		out.println("--------Browser : Servlet5 End --------<br>");
	}
}
```

Servlet6.java:

```
public class Servlet6 extends HttpServlet {
	protected void doGet(HttpServletRequest request,
				HttpServletResponse response)
			throws ServletException ,IOException {

		PrintWriter out = response.getWriter();

		System.out.println("--------Console : Servlet6 Begin--------");
		out.println("--------Browser : Servlet6 Begin --------<br>");

		//Reading Form data by using same request object
		String s3 = request.getParameter("param1");

		System.out.println("Console : Servlet6 : Parameter = " + s3);
		out.println("Browser : Servlet6 : Parameter = " + s3 + "<br>");

		// Reading the revParam from Servlet1 request object
		String revParam3 = (String) request.getAttribute("result");

		System.out.println("Console : Servlet6 : revParam3 = " + revParam3);
		out.println("Browser : Servlet6 : revParam3 = " + revParam3 + "<br>");

		System.out.println("--------Console : Servlet6 End--------");
		out.println("--------Browser : Servlet6 End --------<br>");
	}
}
```

Output :

url :http://localhost:7070/App2/test1.html

url :http://localhost:7070/App2/Servlet4?param1=TIGER

Browser Output:

--------Browser : Servlet4 Begin --------
Browser : Servlet4 : Parameter = TIGER
--------Browser : Servlet5 Begin --------
Browser : Servlet5 : Parameter = TIGER
--------Browser : Servlet6 Begin --------
Browser : Servlet6 : Parameter = TIGER
Browser : Servlet6 : revParam3 = REGIT
--------Browser : Servlet6 End --------
--------Browser : Servlet5 End --------
--------Browser : Servlet4 End --------

Console Output:

```
--------Console : Servlet4 Begin--------
Console : Servlet4 : Parameter = TIGER
--------Console : Servlet5 Begin--------
Console : Servlet5 : Parameter = TIGER
--------Console : Servlet6 Begin--------
Console : Servlet6 : Parameter = TIGER
Console : Servlet6 : revParam3 = REGIT
--------Console : Servlet6 End--------
--------Console : Servlet5 End--------
--------Console : Servlet4 End--------
```

Explanation:

Include() method includes all the Servlets output into the Browser.

What is the difference between include() and forward() ?

The difference is in their name itself.

forward :

- forward is used to forward a request, that is control is transfered to the new servler/jsp. u shud take care to not put any our.println() statements in the servlet from where u plan to call forward method. in fact u cant even fire response.getWriter() method in this case. u can only do that in the servlet to which the control is being forwarded.

- If we use the ***forward()*** method instead of the ***include()*** in our code the **Container** discards the output generated by the **Calling Servlet** and send only the **Called Servlet** response to**Client**. So here we can get only the output of one **Servlet** i.e. **Called Servlet's** output.

- forward is used for roll based Tasks.

include :

- It means that the new servlet/jsp will be procesed and any out.println(). html stuff will be included and then control will come back to the servlet/jsp that called include method.

- If we use ***include()*** method of **RequestDispatcher** to dispatch the request, the **Container** includes the output generated by the **Called Servlet** as part of output generated by **Calling Servlet**. That is we can get outputs of the both **Servlets**.

- Include is used for achieving Reusability. Whatever repeated codes are there, keep that in one Servlet, include that Servlet where ever u want.

sendRedirect

sendRedirect is entirely opposite to RequestDispatcher.

Eg :4

test1.html:

```
<Form action="Servlet7">
        Parameter    :     <inputtype="text"name="param1"/>
                          <inputtype="submit"value="submit"/>
</ Form >
```

Servlet7.java:

```
public class Servlet7 extends HttpServlet {
      protected void doGet(HttpServletRequest request,
                           HttpServletResponse response)
                     throws ServletException, IOException {

                     PrintWriter out = response.getWriter();
                     String s1 = request.getParameter("param1");

                     System.out.println("--------Console : Servlet7 Begin--------");
                     out.println("--------Browser : Servlet7 Begin --------<br>");
                     System.out.println("Console : Servlet7 : Parameter = " + s1);
                     out.println("Browser : Servlet7 : Parameter = " + s1 + "<br>");
                     String revParam = newStringBuffer(s1).reverse().toString();
                     request.setAttribute("result", revParam);
                     response.sendRedirect("Servlet8");
                     System.out.println("--------Console : Servlet7 End--------");
                     out.println("--------Browser : Servlet7 End --------<br>");
      }
}
```

Servlet8.java:

```
public class Servlet8 extends HttpServlet {
      protected void doGet(HttpServletRequest request,
                           HttpServletResponse response)
                     throws ServletException, IOException {
                     PrintWriter out = response.getWriter();
                     System.out.println("--------Console : Servlet8 Begin--------");
                     out.println("--------Browser : Servlet8 Begin --------<br>");
                     //Reading form data
                     String s2 = request.getParameter("param1");
                     System.out.println("Console : Servlet8 : Parameter = " + s2);
                     out.println("Browser : Servlet8 : Parameter = " + s2 + "<br>");

                     response.sendRedirect("Servlet9");

                     System.out.println("--------Console : Servlet8 End--------");
                     out.println("--------Browser : Servlet8 End --------<br>");

      }
}
```

Servlet9.java:

```
Public class Servlet9 extends HttpServlet {
      Protected void doGet(HttpServletRequest request,
                           HttpServletResponse response)
                     throws ServletException ,IOException {

            PrintWriter out = response.getWriter();

            System.out.println("--------Console : Servlet9 Begin--------");
            out.println("--------Browser : Servlet9 Begin --------<br>");

            //Reading Form data by using same request object
            String s3 = request.getParameter("param1");

            System.out.println("Console : Servlet9 : Parameter = " + s3);
            out.println("Browser : Servlet9 : Parameter = " + s3 + "<br>");

            // Reading the revParam from Servlet1 request object
            String revParam3 = (String) request.getAttribute("result");
            System.out.println("Console : Servlet9 : revParam3 = " + revParam3);
            out.println("Browser : Servlet9 : revParam3 = " + revParam3 + "<br>");

            System.out.println("--------Console : Servlet9 End--------");
            out.println("--------Browser : Servlet9 End --------<br>");
      }
}
```

Output :

url :http://localhost:7070/App2/test1.html

Parameter : TIGER Submit → Click

url :http://localhost:7070/Servlet9

Browser Output:

--------Browser : Servlet9 Begin --------
Browser : Servlet9 : Parameter = null
Browser : Servlet9 : revParam3 = null
--------Browser : Servlet9 End --------

Console Output:

```
--------Console : Servlet7 Begin--------
Console : Servlet7 : Parameter = TIGER
--------Console : Servlet7 End--------

--------Console : Servlet8 Begin--------
Console : Servlet8 : Parameter = null
--------Console : Servlet8 End--------

--------Console : Servlet9 Begin--------
Console : Servlet9 : Parameter = null
Console : Servlet9 : revParam3 = null
--------Console : Servlet9 End--------
```

Explanation:

While executing Servlet7,
```
response.sendRedirect("Servlet8");
```

redirection message stored inside the “**response object**”

After executing entire Servlet7, “**response object**” is sending to browser (Redirecting to browser).

Now browser making new request to Servlet8.

While executing Servlet8,
```
response.sendRedirect("Servlet9");
```

redirection message stored inside the “**response object**”

After executing entire Servlet8, “**response**” is sending to browser (Redirecting to browser).

Here also browser making new request to Servlet9.

sendRedirect is happened in browser, Servlet7 sending message to browser, stating that u need to make new request to Servlet8, then from Servlet8 to Servlet9.

Servlet7 is not making request to Servlet8. Browser only making new request to Servlet8.

That is why sendRedirect is called as “**Clientside Technique**”. It is entirely opposite to requestDispatcher. requestDispatcher is a “**Serverside Technique**”.

Whatever “**request object**” we’re using in Servlet7, we’re not using the same “**request object**” in Servlet8 & Servlet9 also. All “**request objects**” are different. Becoz every time browser is making new request to server, so new “**request object**” is created by Server.

Form data we cannot read in Servlet8 & in Servlet9. Becoz Servlet8 & Servlet9, “**request objects**” are different.

sendRedirect is happening with browser intention, So Browser can make request to any Web Application within same Server.

sendRedirect is for within same Application, or any Web Applications within Same Server. But we cannot go for different Servers.

If you like to go for different Servers, then use “**Web Services**”.

Session Management.

See the following Example:

Eg : 5

test1.html:

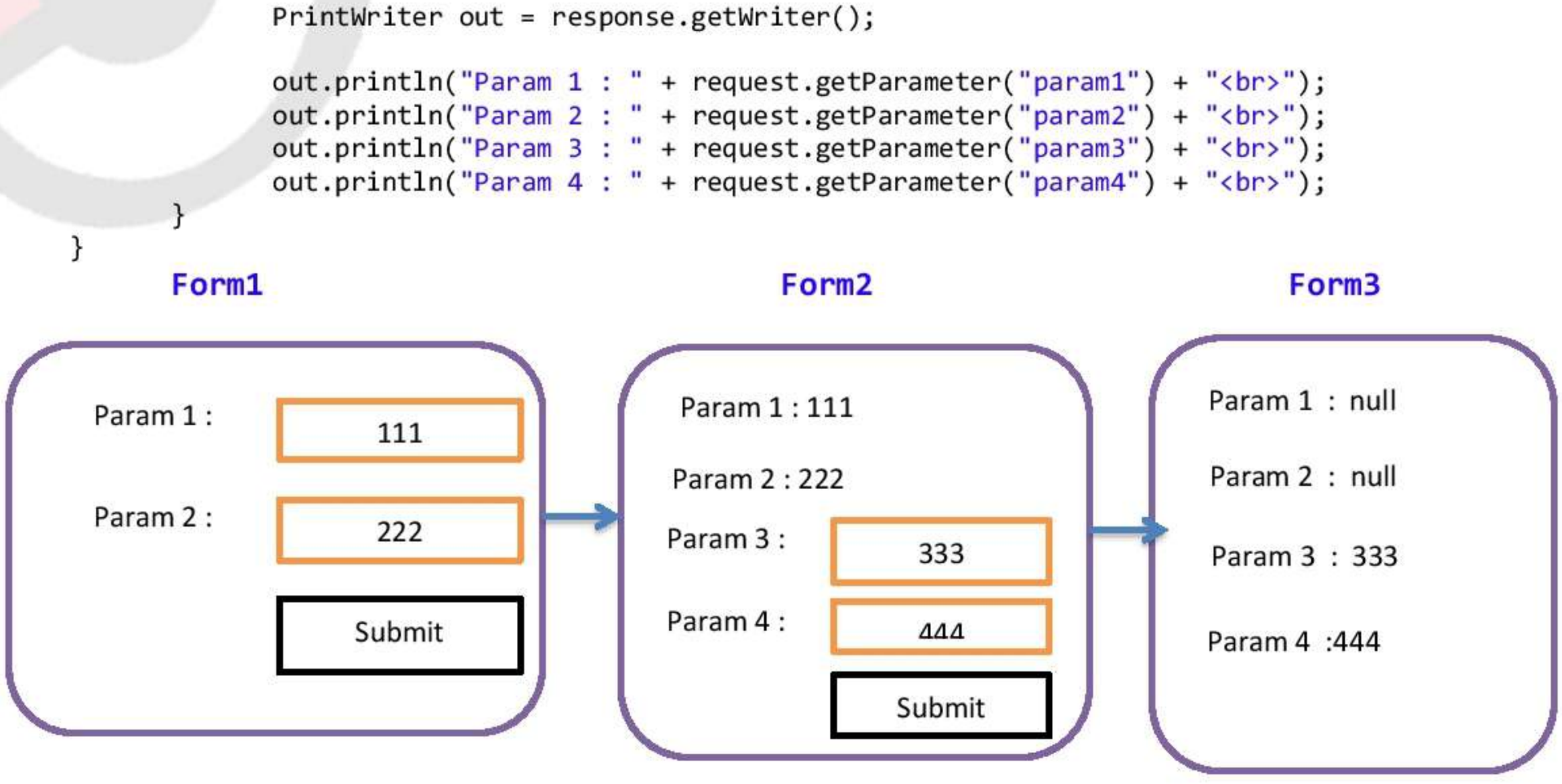
```
<Form action="Servlet3" name = “Form1”>

      Param 1       : <inputtype="text"name="param1"/><br>
      Param 2       : <inputtype="text"name="param2"/><br>
                      <inputtype="submit"value="submit"/>
</Form>
```

Servlet1.java:

```
Public class Servlet1 extends HttpServlet {
      Protected void doGet(HttpServletRequest request,
                           HttpServletResponse response)
                     throws ServletException, IOException {
            PrintWriter out = response.getWriter();

            out.println("Param 1 : " + request.getParameter("param1") + "<br>");
            out.println("Param 2 : " + request.getParameter("param2") + "<br>");

            out.println("<Form action = 'Servlet2' name = 'Form2'>");
            out.println("Param 3 : <input type = 'text' name = 'param3' /><br>");
            out.println("Param 4 : <input type = 'text' name = 'param4' /><br>");
            out.println("<input type = 'submit' value = 'submit' />");
            out.println("</Form>");
      }
}
```

Servlet2.java:

```
public class Servlet2 extends HttpServlet {
      protected void doGet(HttpServletRequest request,
                           HttpServletResponse response)
                     throws ServletException, IOException {

            PrintWriter out = response.getWriter();
            out.println("Param 1 : " + request.getParameter("param1") + "<br>");
            out.println("Param 2 : " + request.getParameter("param2") + "<br>");
            out.println("Param 3 : " + request.getParameter("param3") + "<br>");
            out.println("Param 4 : " + request.getParameter("param4") + "<br>");
      }
}
```



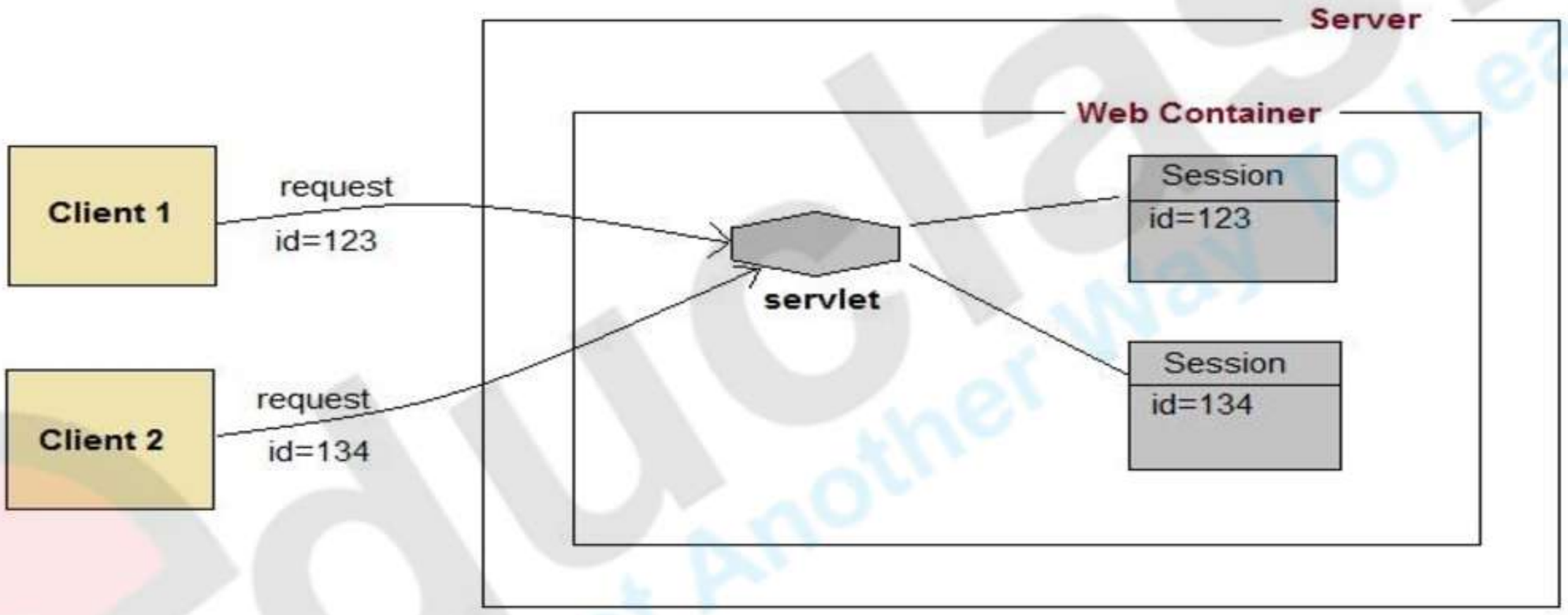
What is a session?

HTTP protocol and Web Servers are stateless, what it means is that for web server every request is a new request to process and they can't identify if it's coming from client that has been sending request previously.

We all know that **Http** is a stateless protocol. Each request and response is independent. But somehow you need to keep track of client activity across multiple request i.e storing session information for a particular client.

When a client sends a request, the server sends back a response but does not keep any information about that requestand the client state.
In most web applications a client has to access various pagesbefore completing a specific task and the client state shouldbe kept along all those pages

How sessions work



The techniques for managing the state of an end user are:

1. Hidden form field
2. URL rewriting
3. Cookies
4. Servlet session API [HttpSession]

1. Hidden form field:

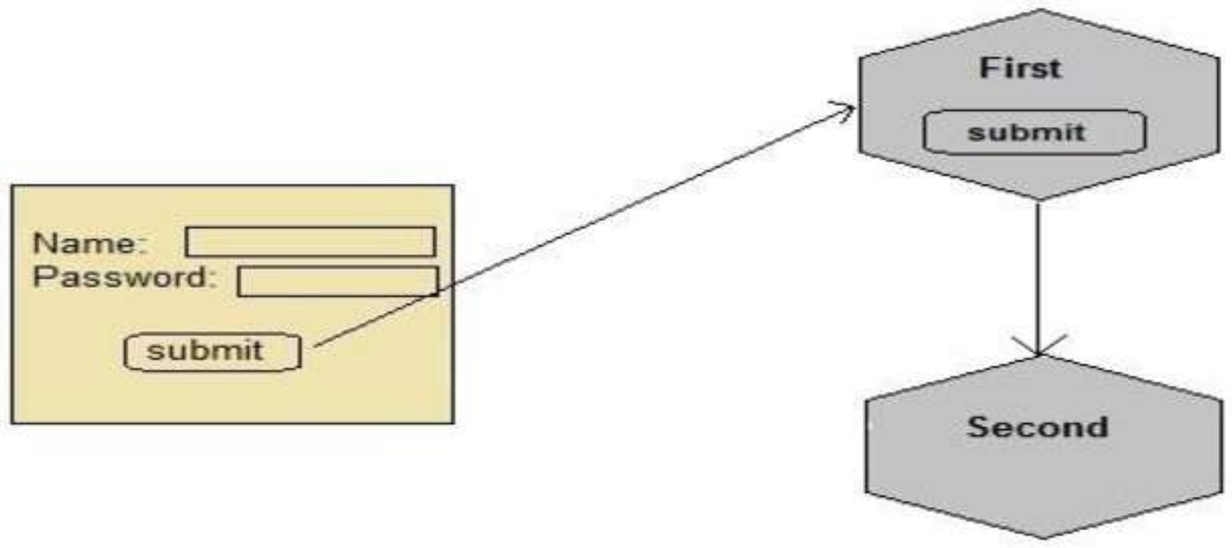
- Simplest technique to maintain the state of an end user.
- Embedded in an HTML form.
- The client state is passed from the server to the client and back to the server in a hidden field of a form
- Hidden form field can also be used to store session information for a particular client. In case of hidden form field a hidden field is used to store client state. In this case user information is stored in hidden field value and retrieve from another servlet.
- Not visible when you view an HTML file in a browser window.
- Not able to maintain the state of an end user when it encounters a static document.

Advantage

- Does not have to depend on browser whether the cookie is disabled or not.

Disadvantage

- Extra form submission is required on every page.



Eg :6

test1.html:

```
<Formaction="Servlet3" name = “Form1”>

      Param 1      : <inputtype="text"name="param1"/><br>
      Param 2      : <inputtype="text"name="param2"/><br>
                     <inputtype="submit"value="submit"/>
</Form>
```

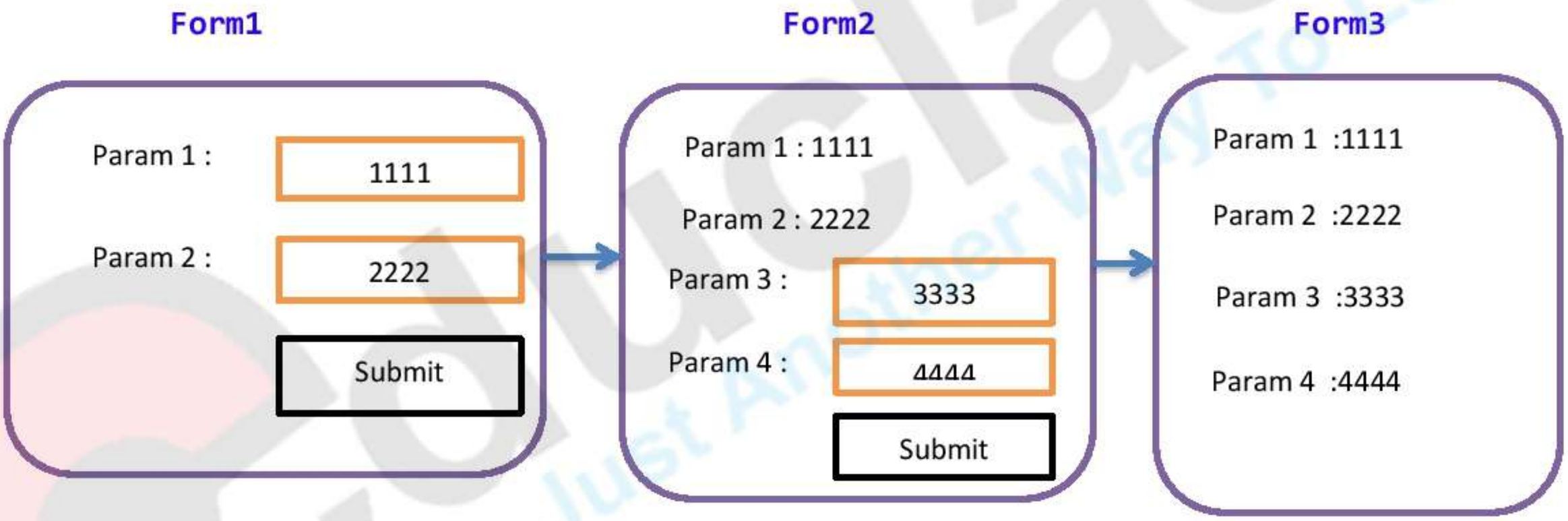
Servlet3.java:

```
public class Servlet3 extends HttpServlet {
      protected void doGet(HttpServletRequest request,
                           HttpServletResponse response)
                     throws ServletException, IOException {

            PrintWriter out = response.getWriter();

            String s1 = request.getParameter("param1");
            String s2 = request.getParameter("param2");
            out.println("Param 1 : " + s1 + "<br>");
            out.println("Param 2 : " + s2 + "<br>");

            out.println("<Form Action = 'Servlet4' name = 'Form 2' method = 'post'>");

            // Adding 2 Hidden Fields.
            out.println("<input type = 'hidden' name = 'param1' value = '"+s1+"'/> ");
            out.println("<input type = 'hidden' name = 'param2' value = '"+s2+"'/> ");

            out.println("Param 3 : <input type = 'text' name = 'param3' /><br>");
            out.println("Param 4 : <input type = 'text' name = 'param4' /><br>");
            out.println("<input type = 'submit' value = 'submit' />");
            out.println("</Form>");
      }
}
```

Servlet4.java:

```
public class Servlet4 extends HttpServlet {
      protected void doGet(HttpServletRequest request,
                           HttpServletResponse response)
                     throws ServletException, IOException {

            PrintWriter out = response.getWriter();

            out.println("Param 1 : " + request.getParameter("param1") + "<br>");
            out.println("Param 2 : " + request.getParameter("param2") + "<br>");

            out.println("Param 3 : " + request.getParameter("param3") + "<br>");
            out.println("Param 4 : " + request.getParameter("param4") + "<br>");

      }
}
```

Output:



http://www.javatpoint.com/jsp-tutorial

2. URL Rewriting

- Maintains the state of end user by modifying the URL.
- Is used when the information to be transferred is not critical.
- Cannot be used for maintaining the state of an end user when a static document is encountered.
- User can't modify the URL.

Explanation:

The client state is passed from the server to the client andback to the server in the query string, accompanying theURL

In URL rewriting, we append a token or identifier 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, a parameter 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.

Advantage of URL Rewriting

1. It will always work whether cookie is disabled or not (browser independent).
2. Extra form submission is not required on each pages.

Disadvantage of URL Rewriting

1. It will work only with links.
2. It can send Only textual information.

Eg :7

test1.html:

```
<Formaction="Servlet5" name = “Form1”>

      Param 1      : <inputtype="text"name="param1"/><br>
      Param 2      : <inputtype="text"name="param2"/><br>
                     <inputtype="submit"value="submit"/>
</Form>
```

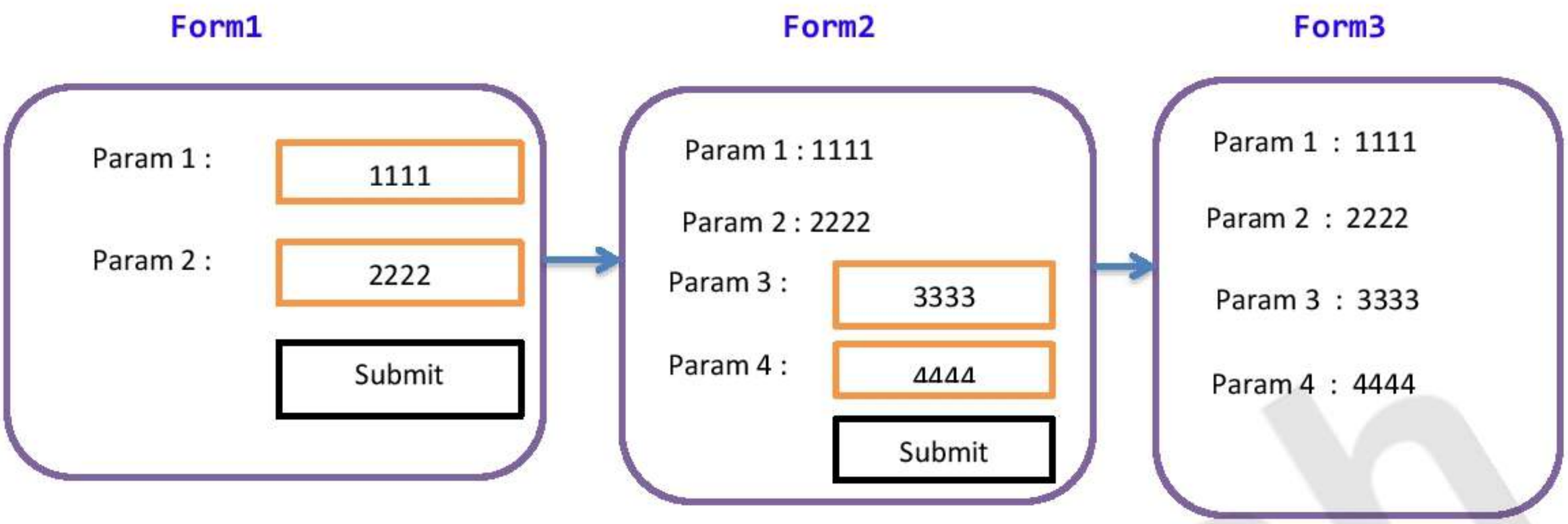
Servlet5.java:

```
public class Servlet5 extends HttpServlet {
      @Override
      Protected void doGet(HttpServletRequest req,
                           HttpServletResponse res)
                     throws ServletException, IOException {

            String s1 = req.getParameter("param1");
            String s2 = req.getParameter("param2");

            PrintWriter out = res.getWriter();

            out.println("Param 1 : " + s1 + "<br>");
            out.println("Param 2 : " + s2 + "<br>");

            String newUrl = "Servlet6?param1= "+s1+"&param2="+s2;

            out.println("<Form action = '" +newUrl+"' name = 'Form1' method = 'post'>");
            out.println("Param 3 : <input type = 'text' name = 'param3' /><BR>");
            out.println("Param 4 : <input type = 'text' name = 'param4' /><BR>");
            out.println("<input type = 'submit' value = 'submit' /> ");
            out.println("</Form> ");
      }
}
```

Servlet6.java:

```
public class Servlet6 extends HttpServlet {
      @Override
      Protected void doPost(HttpServletRequest req,
                            HttpServletResponse res)
                     throws ServletException, IOException {

            PrintWriter out = res.getWriter();

            out.println("Param 1 : " + req.getParameter("param1") + "<BR>");
            out.println("Param 2 : " + req.getParameter("param2") + "<BR>");
            out.println("Param 3 : " + req.getParameter("param3") + "<BR>");
            out.println("Param 4 : " + req.getParameter("param4") + "<BR>");
      }
}
```

Output:



Eg :8

test1.html:

```
<form action="Servlet7">
	Name   :	<inputtype="text"name="userName"/><br/>
			<inputtype="submit"value="Go"/>
</form>
```

Servlet5.java:

```
public class Servlet7 extends HttpServlet {

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

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

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

			//appending the username in the query string
			out.print("<a href='Servlet8?uname="+user+"'>"
					+ "Click here to Visit</a>");

			out.close();

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

Servlet6.java:

```
public class Servlet8 extends HttpServlet {

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

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

		//getting value from the query string
		String user = request.getParameter("uname");
		out.print("Hello "+ user);

		out.close();
	}
}
```

Output: