



**Softra Services**

Learn Today, Lead Tomorrow

Advanced Java

Servlets and JSPs

# Agenda

- Understand and Create Web Application.
- Explain Servlet and its life cycle.
- Define Servlet Context and Servlet Config
- Understand and create Java server pages
- Explain JSP and JSP lifecycle
- Defining JSP scripting elements and implicit objects
- Introduction to MVC

# Topic List

**Introduction to Servlet**

**Servlet Life Cycle**

**Deployment Descriptor**

**RequestDispatcher**

**Servlet Config and Servlet Context**

# Topic List

**Introduction to JSP**

**JSP Lifecycle**

**JSP Scripting elements**

**JSP Implicit objects**

**Model-View-Controller (MVC)**

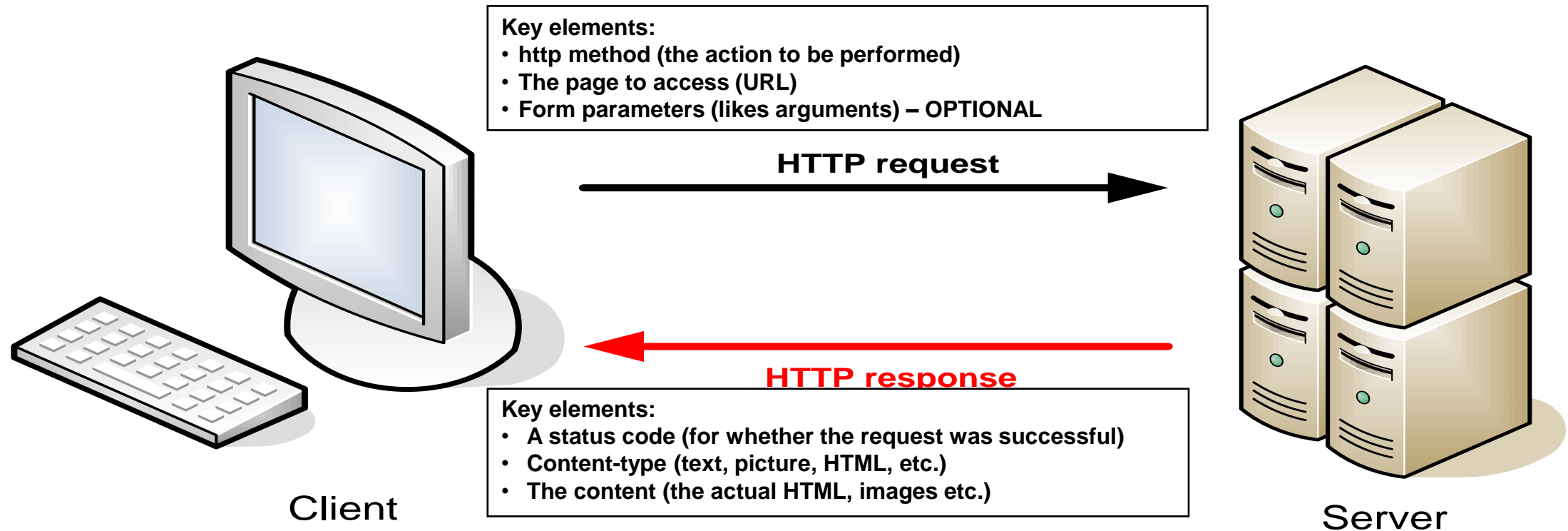
# Introduction to Servlet

- Web Application
- A Web application (Web app) is an application program that runs on a web / application server and can be accessed over the Internet through a web browser.
- When the client tries to access a web site, browser sends the request to the corresponding web server and the server will send the requested webpage and the related files to the client back.



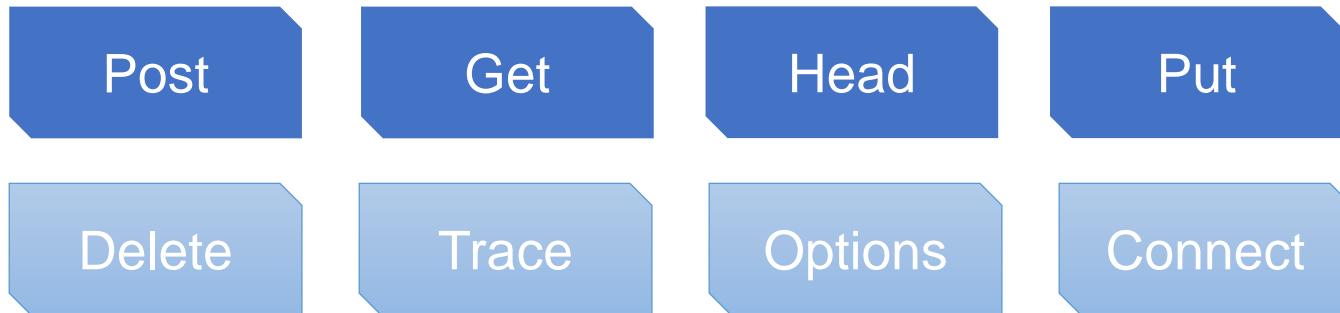
# Introduction to Servlet

- Hypertext Transfer Protocol(HTTP)
- used to transfer instructions and data between machines over the web.
- uses a server-client model. A client can be a computer, laptop, or mobile device. HTTP server can be Apache or JBOSS.
- Deliver/access World Wide Web data and files.



# Introduction to Servlet

- HTTP Methods
- There are number of HTTP methods. GET and POST are the commonly used methods.



- **POST** - Submits data to be processed (e.g. from an HTML form) to the identified(specified) resource. The data is included in the body of the request.
- **GET** - Requests a representation of the specified resource. Get should not be used for operations that cause side-effects such as taking actions on web pages.

# Introduction to Servlet

- Difference between GET and POST methods

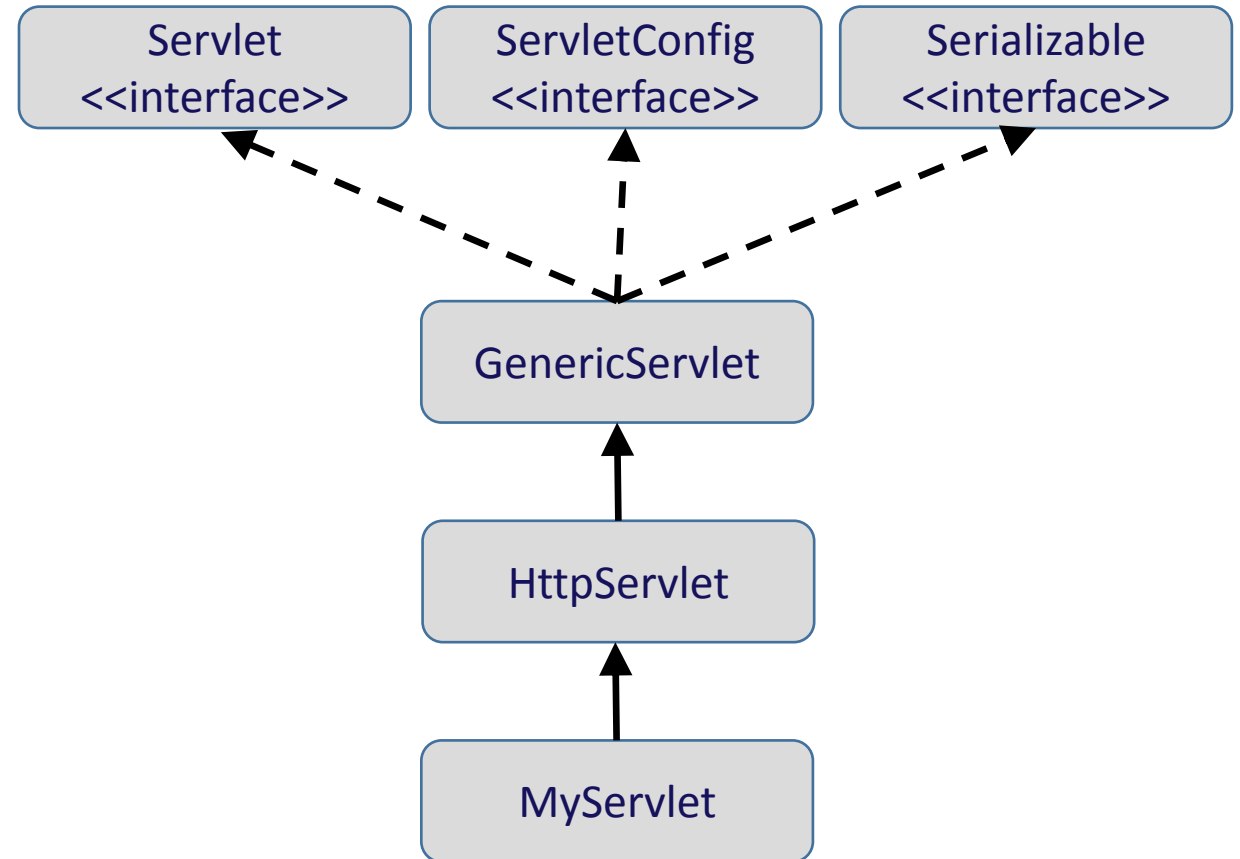
GET Method	POST method
GET request contains only request line and HTTP header.	POST request contains HTTP body along with the request line and header.
Parameters are appended to the URL and sent along with header information.	Parameters are not appended to the URL instead passed as a part of HTTP body. This method is used whenever we need to send some sensitive information to server.
Data to be passed is limited. We can send only small amount of data.	we can send a huge amount of data.
Example: <a href="http://localhost:8082/Module5Demo/login?uname=John&amp;pwd=John">http://localhost:8082/Module5Demo/login?uname=John&amp;pwd=John</a>	Example: <a href="http://localhost:8082/Module5Demo/login">http://localhost:8082/Module5Demo/login</a>



# Introduction to Servlet

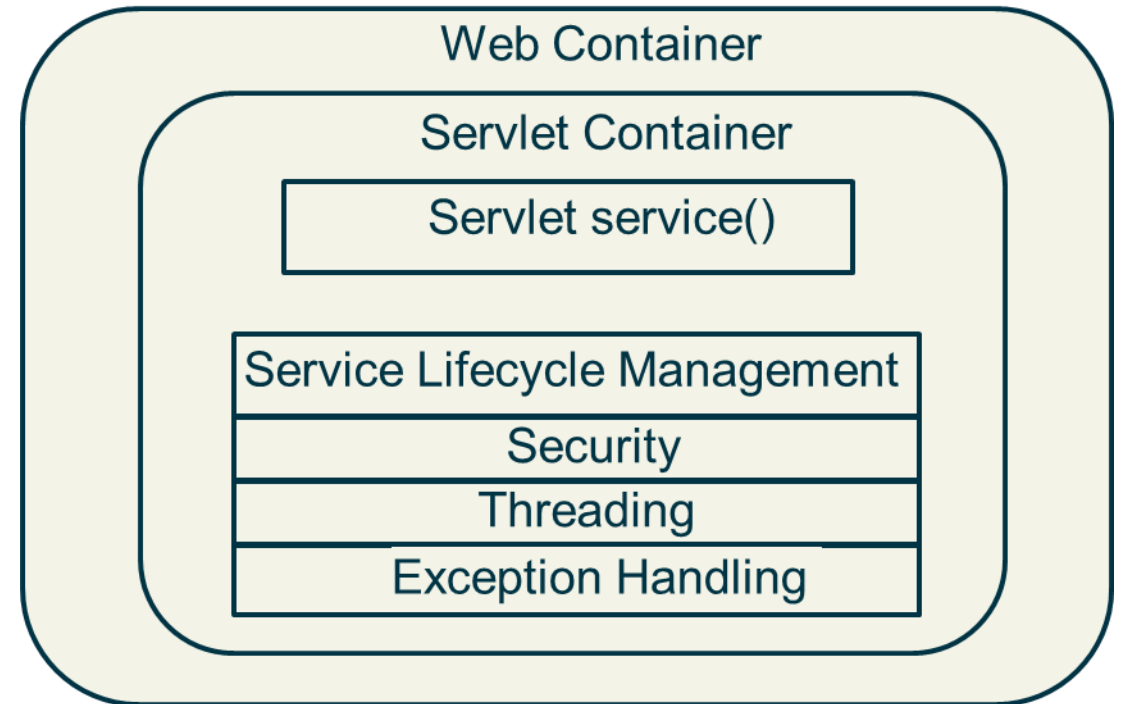
- What is Servlet?
- server-side java program.
- used for processing web-based HTTP client requests.
- belongs to only one application and can be concurrently accessed by multiple requests.
- runs in a Servlet container under the Application Server (Tomcat).
- GenericServlet is an abstract class and it's a protocol-independent servlet.
- HttpServlet is an abstract class, defines a HTTP protocol specific servlet.
- MyServlet is a user-defined class extends HttpServlet. Overrides doGet() / doPost() based on client request method.

## Servlet API



# Introduction to Servlet

- Web Container
- Web Container is a Java runtime environment (JRE) which is responsible for managing the life cycle of JSP pages and Servlets.
- It is part of the web server or application server that provides the network services over which requests and responses are sent.
- Servlet container implements the Java Servlet specification and resides in an application server.



# Servlet Life Cycle

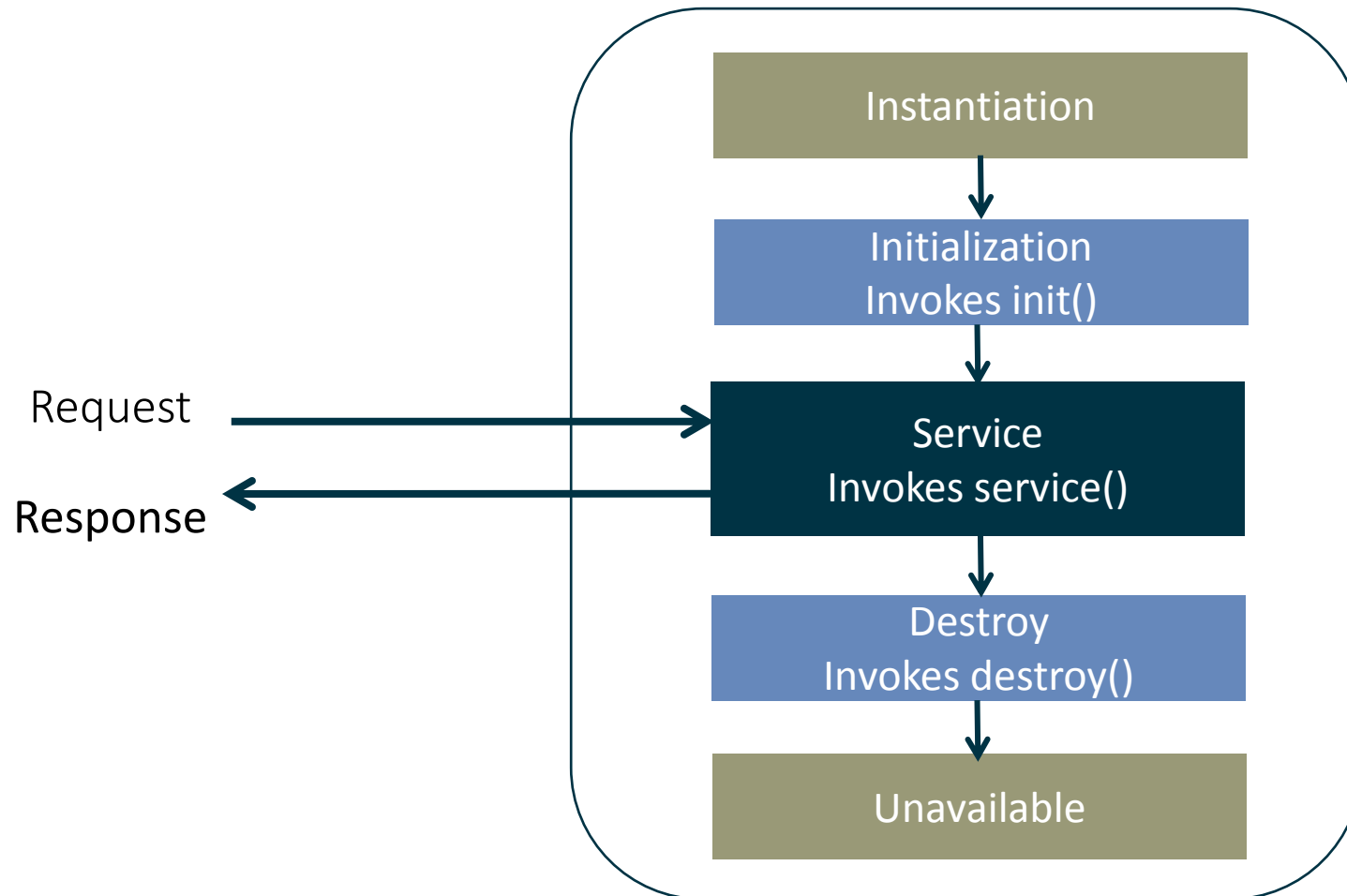
- Servlet Life Cycle Methods

When a client request is made for a particular Servlet, Web Container performs the following steps:

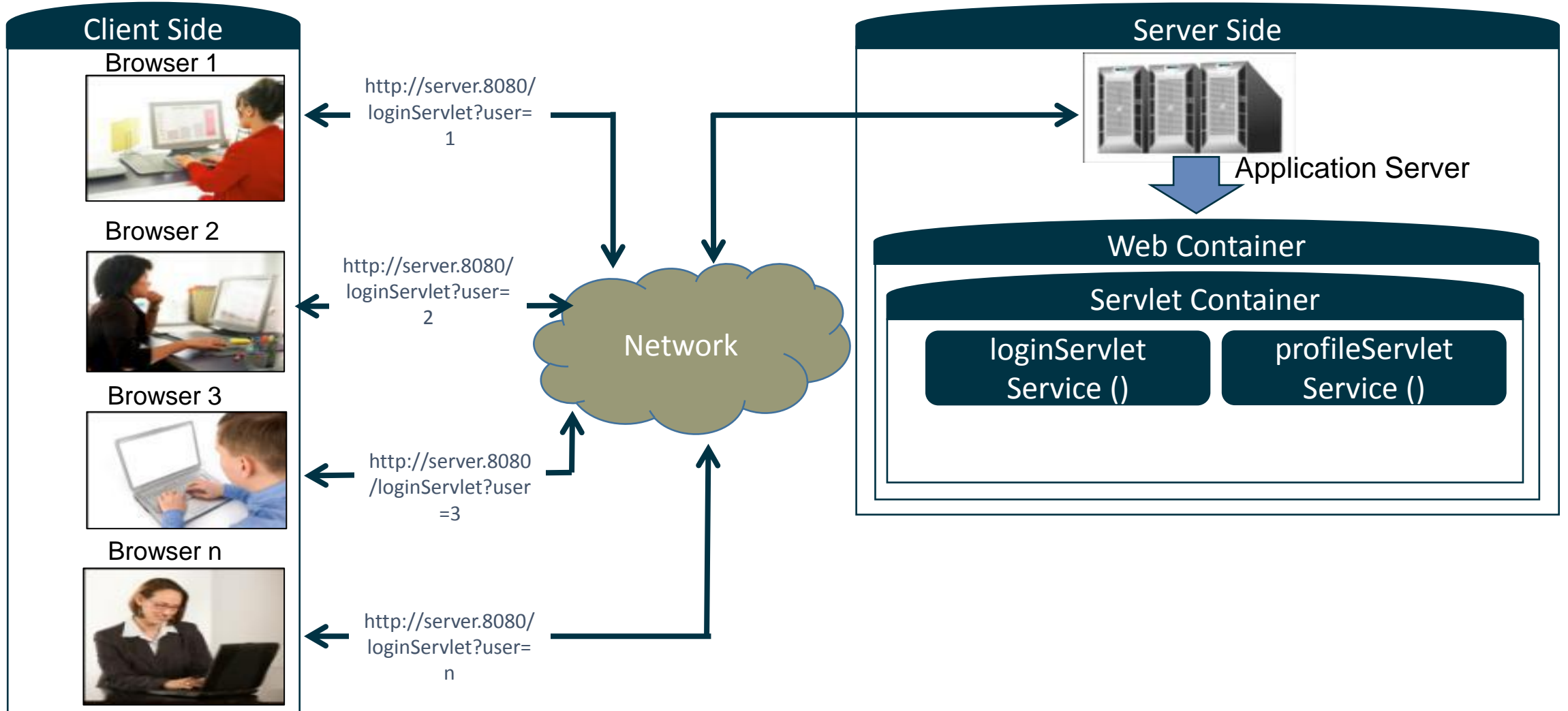
- Step 1: Load the servlet class.
- Step 2: Create an instance of the servlet class.
- Step 3: Initializes the servlet instance.
  - Init() method is called only once, when a very first request comes to a servlet.
  - Container calls the init() method before the servlet processes any client request.
- Step 4: service() method is where the actual task of the servlet is defined.
  - Invokes the service() method every time a request comes for a servlet.
  - It has 2 parameters - request and response objects.
  - It determines the Http method(GET/POST) based on the client request and calls either doGet() / doPost() accordingly.
- Step 5: Invoke the destroy() method, when a servlet instance is no longer required.

# Servlet Life Cycle

- Servlet Life Cycle Methods

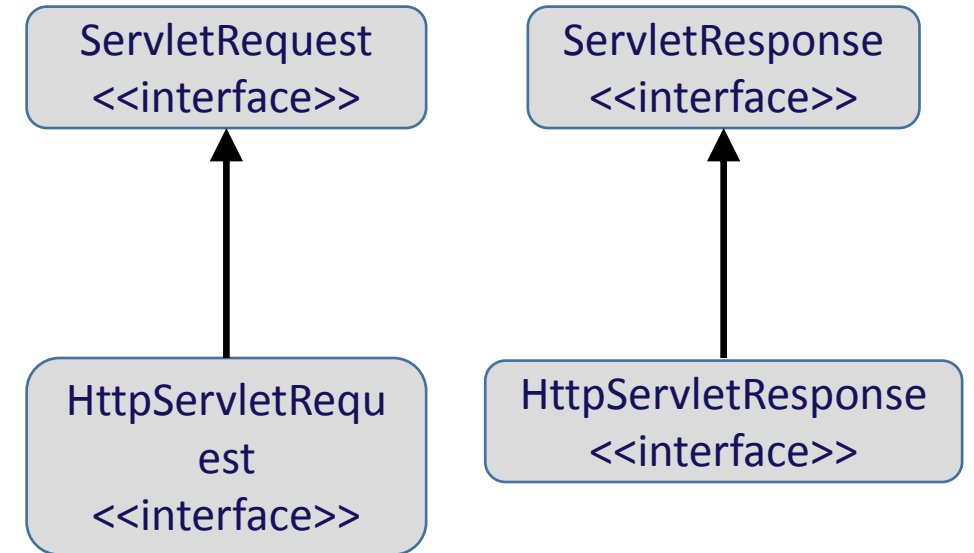


# Servlet Life Cycle



# Servlet Life Cycle

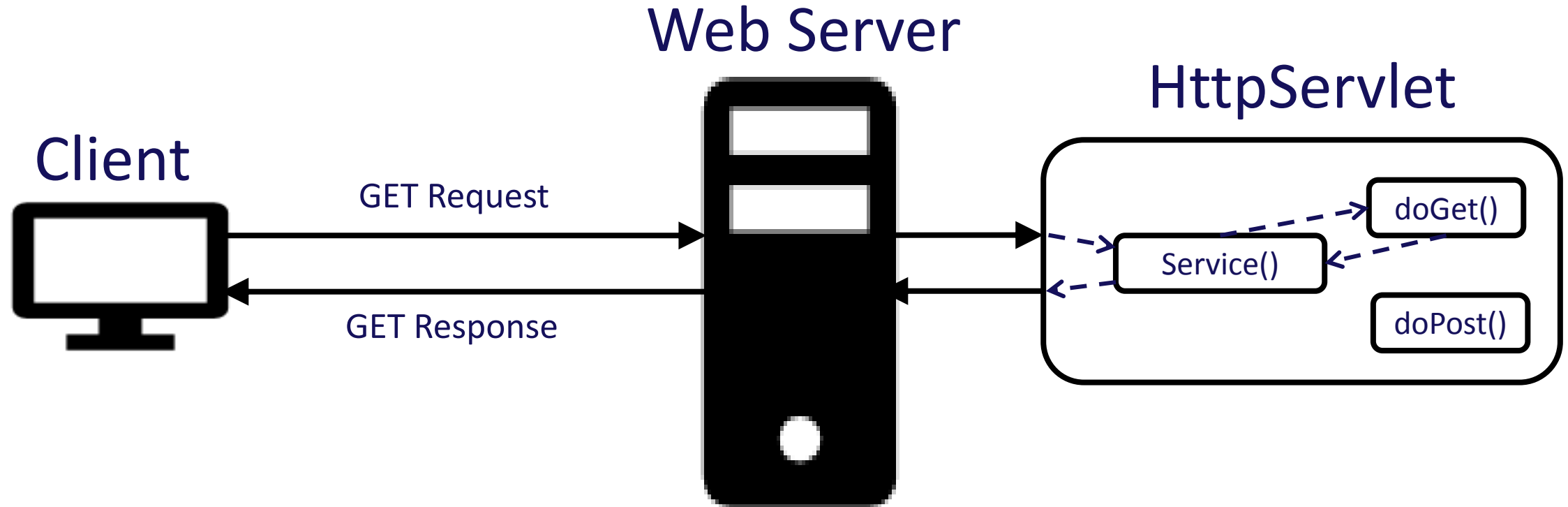
- HttpServletRequest and HttpServletResponse
- **HttpServletRequest** is called as Request object.
- It contains information about the request made by the client.
- String getParameter(String name) in the ServletRequest is used to retrieve the value of the request parameter. Returns null if the parameter doesn't exist.
- String[] getParameterValues(String name) returns String array containing all the values of the given parameter. Returns null if the parameter doesn't exist.
- **HttpServletResponse** is called as Response object.
- Used to send the response information.
- void setContentType(String type) used to set the content type of the response being sent to the client.
- PrintWriter getWriter() is used to write the response body.



- Note: Both these objects are sent as parameters to doGet() and doPost() methods.

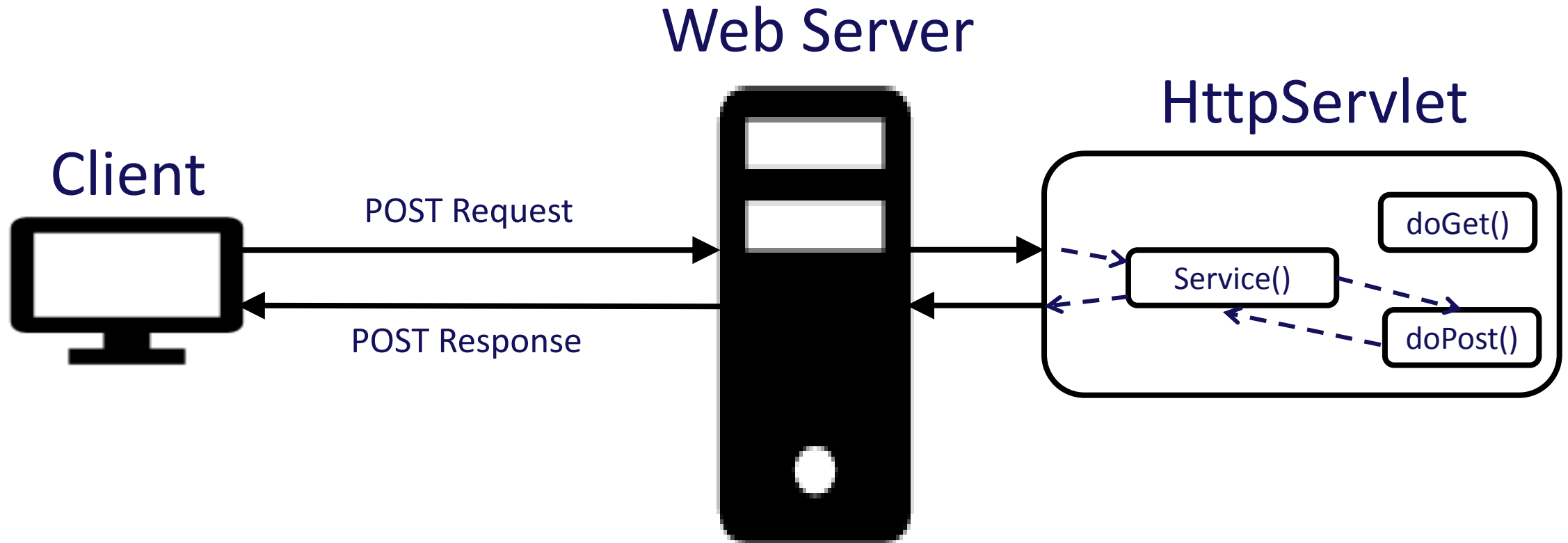
# Servlet Life Cycle

- `doGet(HttpServletRequest request, HttpServletResponse response)`



# Servlet Life Cycle

- `doPost(HttpServletRequest request, HttpServletResponse response)`





# Deployment Descriptor

- Deployment Descriptor(web.xml)
- A web application's deployment descriptor describes the classes, resources and configuration of the application and how the web server uses them to serve web requests.
- When the web server receives a request for the application, it uses the deployment descriptor to map the URL of the request to the code that ought to handle the request.
- The deployment descriptor is named as web.xml.
- It resides in the app's WAR under the WEB-INF/ directory.

## Example

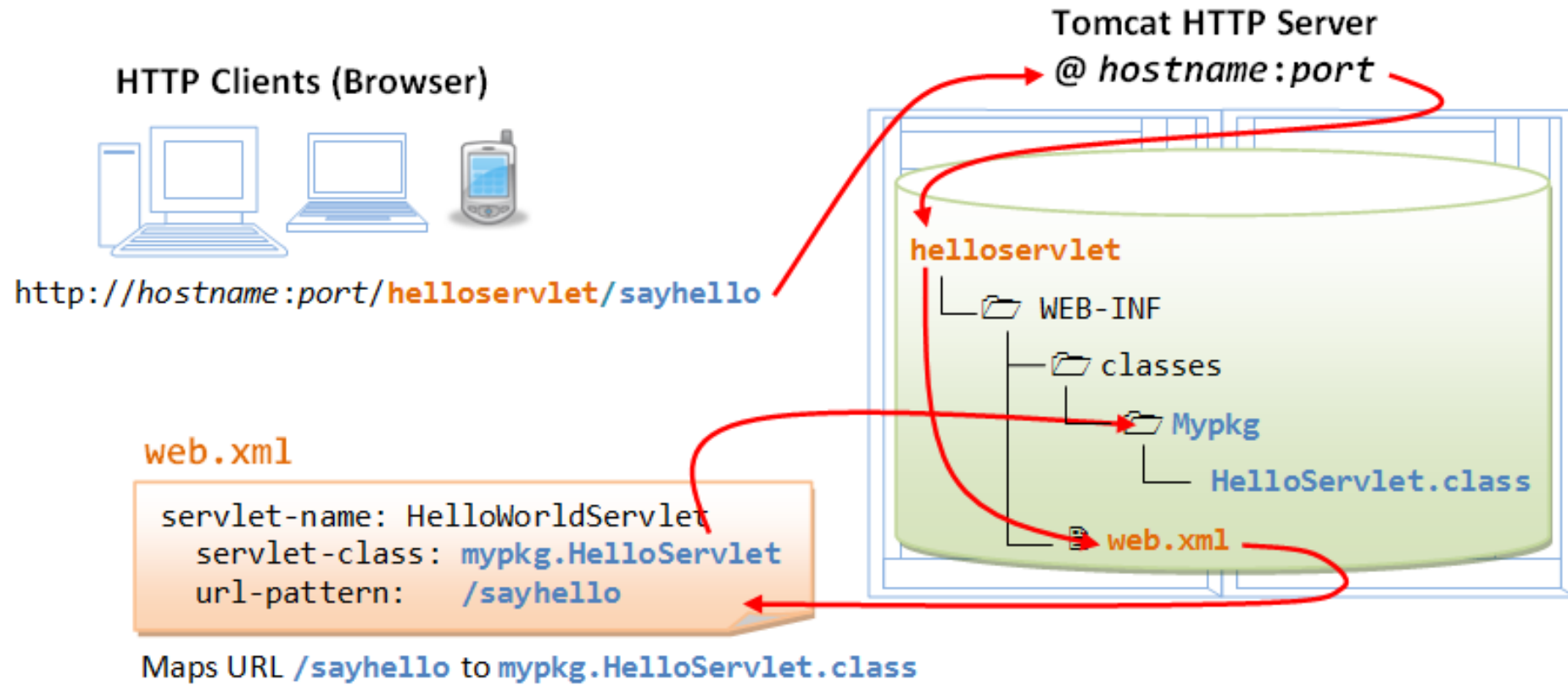
```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
version="2.5">

<servlet>

<servlet-name>HelloWorldServlet</servlet-name>
<servlet-class>mypkg.HelloServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name> HelloWorldServlet</servlet-name>
<url-pattern>/sayhello</url-pattern>
</servlet-mapping>
</web-app>
```

# Deployment Descriptor



# Deployment Descriptor

- @WebServlet Annotation
- In Servlet 3.0, servlet mapping can be done using Annotations.
- @WebServlet annotation is the replacement of servlet configuration in web.xml.

## Code in Web.xml

```
<servlet>
<servlet-name>HelloWorldServlet</servlet-name>
<servlet-class> mypkg.HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name> HelloWorldServlet</servlet-name>
<url-pattern>/sayhello</url-pattern>
</servlet-mapping>
```

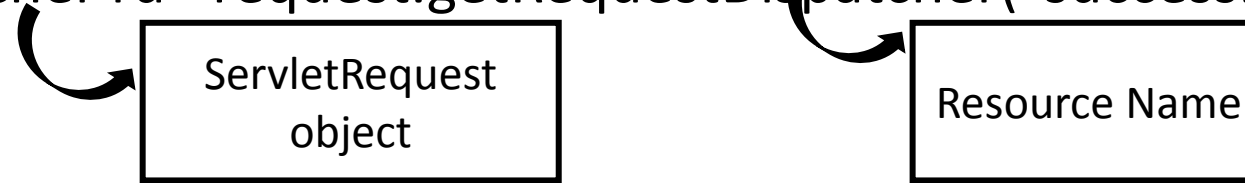
## Servlet Class

```
@WebServlet(name=" HelloWorldServlet ",
urlPatterns={"/sayhello"})
public class HelloServlet extends HttpServlet{
//code
}
```

# RequestDispatcher

- What is RequestDispatcher? And How to get it?
- RequestDispatcher is an interface, defines an object that receives the client request and sends it to any resource(Servlet/JSP/HTML) on the server.
- `getRequestDispatcher()` method is used to get the object of RequestDispatcher.

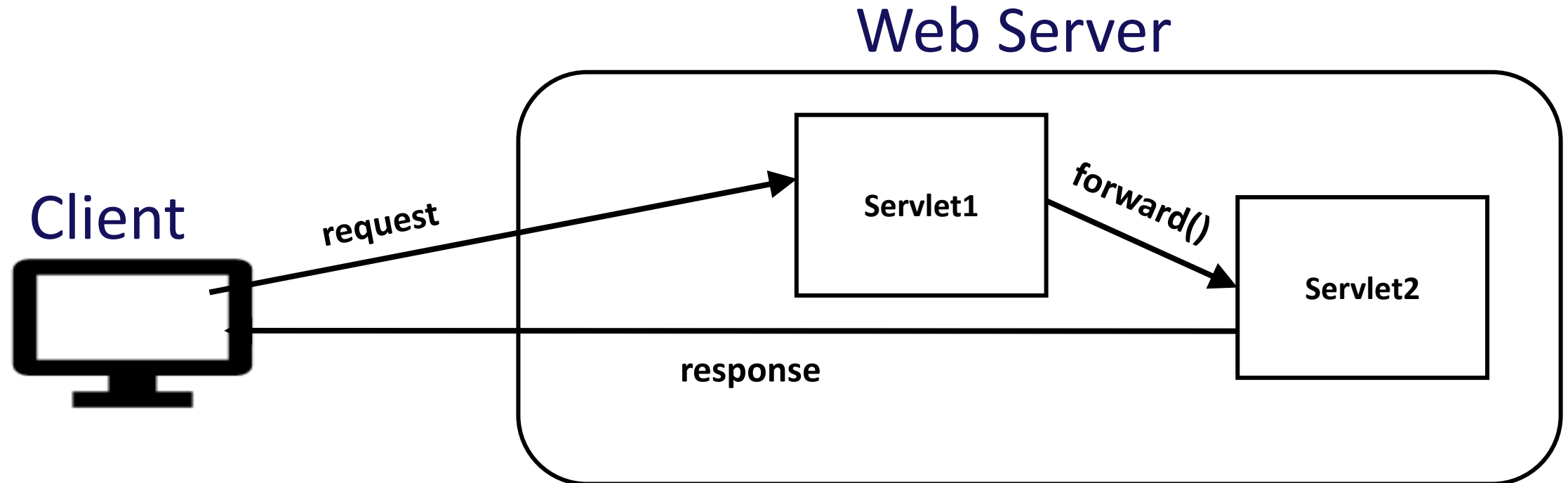
```
RequestDispatcher rd= request.getRequestDispatcher("success.html");
```



- It has 2 methods
  - `forward(ServletRequest request, ServletResponse response)` – forwards the request from a servlet to any other resource on the server whose response will be sent to the user.
  - `include(ServletRequest request, ServletResponse response)` – instead of forwarding the request to another resource this includes the contents of a resource in the response.

# RequestDispatcher – forward()

- RequestDispatcher – forward() method

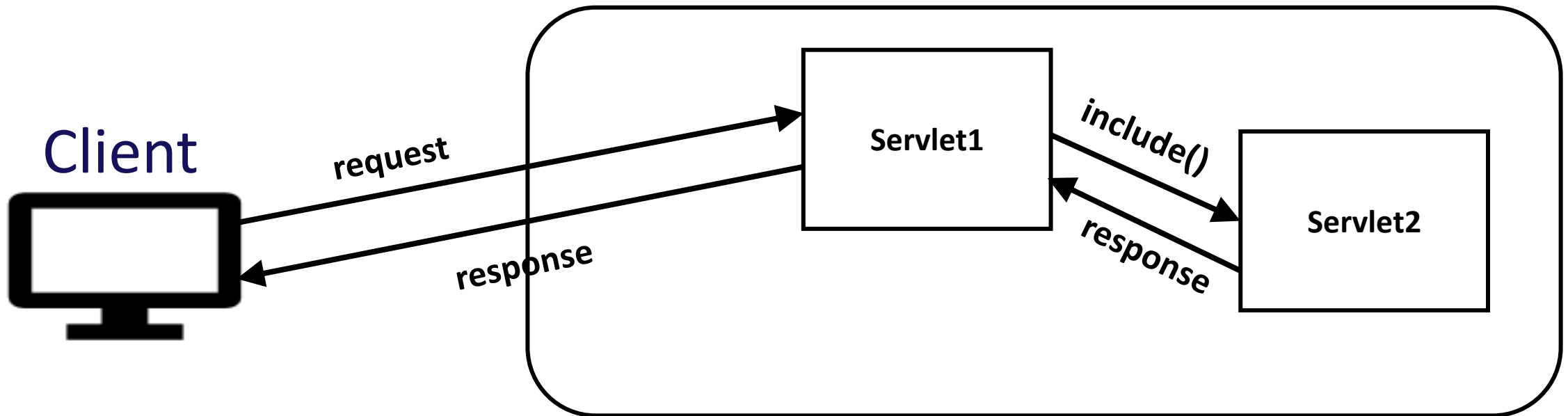


Response of only Servlet2 is sent to the client.

# RequestDispatcher – include()

- RequestDispatcher – include() method

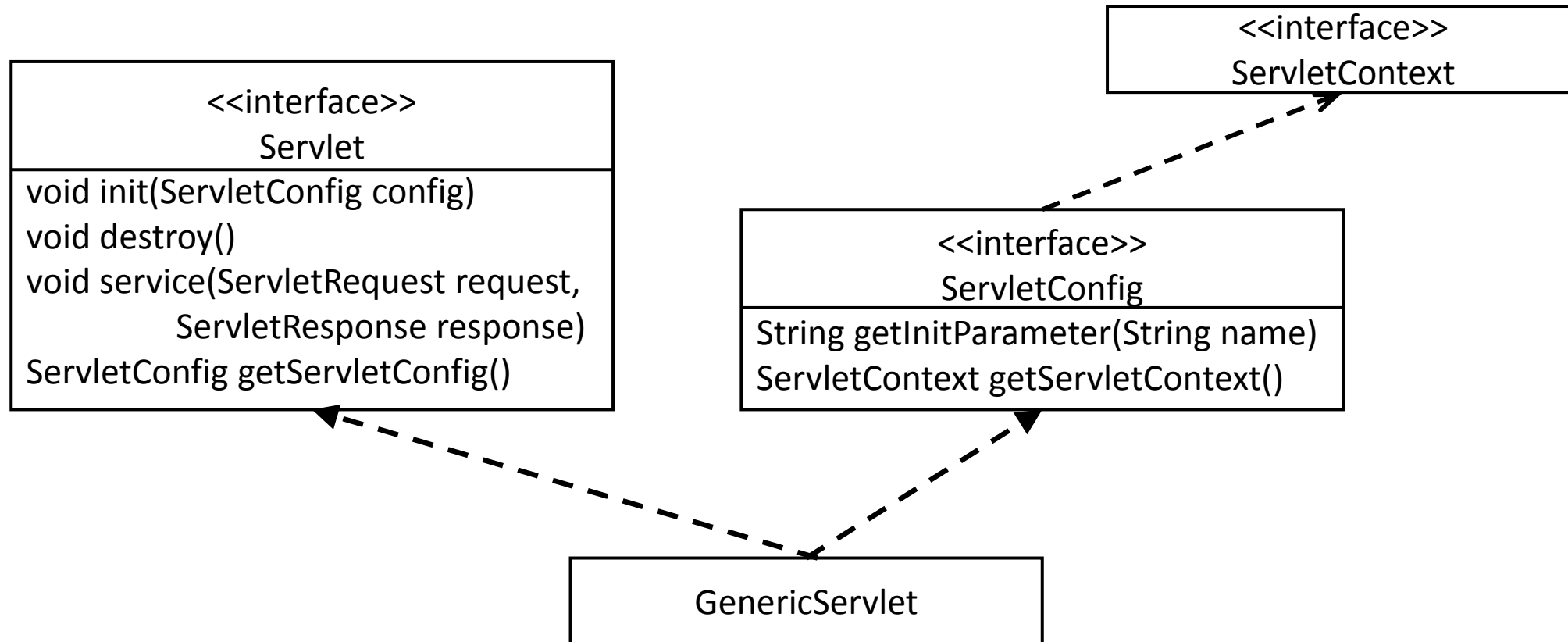
## Web Server



Response of both Servlet1 and Servlet2 are sent to the client.

# Servlet Config and Servlet Context

- Servlet API



# ServletConfig and ServletContext

- Config and Context Object
- In Servlet , Config and context objects are used to access information that are specified in web.xml.
- These interfaces are available in javax.servlet package
- These elements contains the initialization parameters that can be shared across multiple servlets or specific to a particular servlet.
- Objects of these interface access the `getInitParameter()` method which can be used to access initialization parameters.



# ServletConfig Interface

- ServletConfig Object
- This is an interface where the object of this Config is used by the container to redirect data to a servlet.
- This is specific to the particular servlet where the parameter is configured.
- There can be many ServletConfig for a web application. Every Servlet can have its own Configuration information.
- These initialization parameters are specified using <init-param> element in web.xml

```
<Servlet>
```

```
    <init-param>
```

```
        <param-name>parameterName</param-name>
```

```
        <param-value>parameterValue</param-value>
```

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

```
</Servlet>
```

# ServletContext Interface

- ServletContext Object
- ServletContext is an interface which contains list of abstract methods.
- These methods are used by the servlet to communicate with the servlet container.
- This object is used to access the configurations that are specified in deployment descriptor (web.xml)
- This Context object is used to establish communication between applications.
- There should be only one ServletContext for every web application.
- These initialization parameters can be specified using <context-param> element in web.xml.
- These configurations are common and accessible by all the servlet present in the web application.

# ServletContext Interface

- Methods to access Context object
- There are two ways to access ServletContext Object.
  - using `getServletContext()` method present in `ServletConfig` interface.
  - using `getServletContext()` method present in `GenericServlet` class.
- In `web.xml` configuration can be given as

```
<web-app>
    <context-param>
        <param-name>parameterName</param-name>
        <param-value>parameterValue</param-value>
    </context-param>
</web-app>
```

# Introduction to JSP

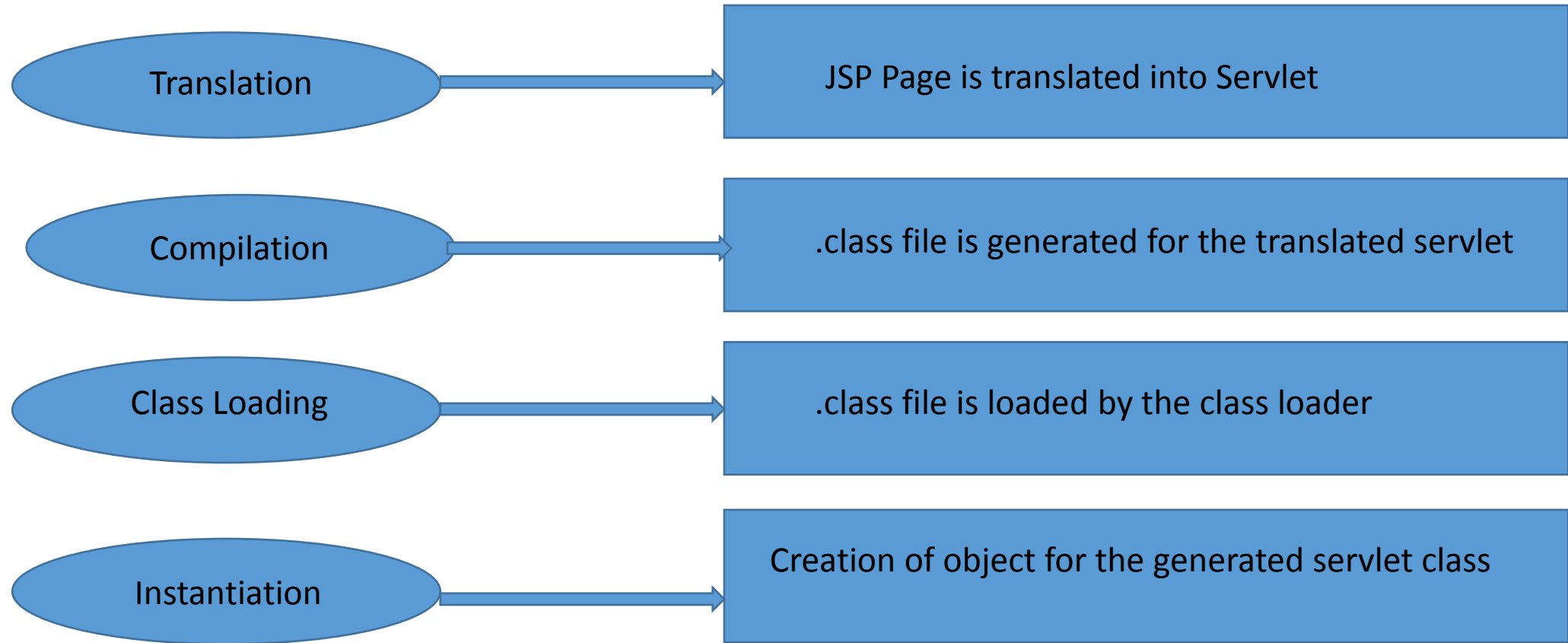
- What is JSP?
- JSP is used to create dynamic web pages in web applications
- It consist of HTML elements and JSP elements together.
- Many features can be provided in JSP like Expression Language, Tag libraries, JSP actions etc. which helps in separation of design and development codes
- In short JSP = HTML + Java
- i.e. it contains Java code embedded in an HTML page.

# Introduction to JSP

- Why JSP?
- JSP is recommended for several application development because of its rich features. Some of the features are
  - More features embedded than Servlet
  - Good Separation of design and development code
  - When there is any change, recompilation efforts are less in JSP.
  - Reduces much Java code in presentation layer.

# JSP Lifecycle

- JSP Lifecycle
- Following are the various phases present in lifecycle of a JSP page.



# JSP Lifecycle

- JSP and translated Servlet

```
Sample.jsp
<%@ page language="java" import="java.util.List"
contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://ww
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<!-- Declaration --%>
<%! String track; %>
<!-- Scriptlet --%>
<% track = "Digital"; %>
<!-- Expression --%>
<%= track %>
</body>
</html>
```

```
public void _jspInit() {
}

public void _jspDestroy() {
}

public void _jspService(final javax.servlet.http.HttpServletRequest request, final javax.servlet.http.HttpServletResponse response)
throws java.io.IOException, javax.servlet.ServletException {

    final javax.servlet.jsp.PageContext pageContext;
    javax.servlet.http.HttpSession session = null;
    final javax.servlet.ServletContext application;
    final javax.servlet.ServletConfig config;
    javax.servlet.jsp.JspWriter out = null;
    final java.lang.Object page = this;
    javax.servlet.jsp.JspWriter _jspx_out = null;
    javax.servlet.jsp.PageContext _jspx_page_context = null;

    try {
        response.setContentType("text/html; charset=ISO-8859-1");
        pageContext = _jspxFactory.getPageContext(this, request, response,
            null, true, 8192, true);
        _jspx_page_context = pageContext;
        application = pageContext.getServletContext();
        config = pageContext.getServletConfig();
        session = pageContext.getSession();
        out = pageContext.getOut();
        _jspx_out = out;

        out.write("\r\n");
        out.write("<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">\r\n");
        out.write("<html>\r\n");
        out.write("<head>\r\n");
        out.write("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=ISO-8859-1\">\r\n");
        out.write("<title>Insert title here</title>\r\n");
        out.write("</head>\r\n");
        out.write("<body>\r\n");
        out.write(' ');
        out.write(' ');
        out.write(' ');
    }
}
```

# JSP Lifecycle

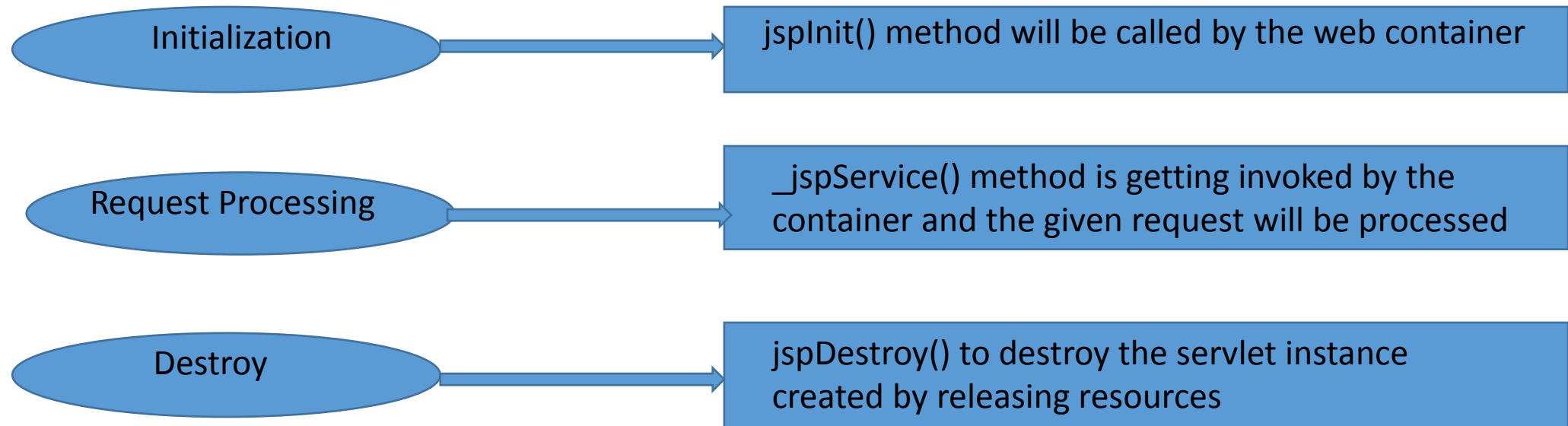
- JSP and translated Servlet
- The generated servlet can be found in workspace metadata folder

This PC > Documents > Eclipse workspace > .metadata > .plugins > org.eclipse.wst.server.core > tmp0 > work > Catalina > localhost > Module5Demo6 > org > apache > jsp				
	Name	Date modified	Type	Size
Sample_jsp.class	12/27/2017 11:35 ...	CLASS File	5 KB	
Sample_jsp	12/27/2017 11:35 ...	JAVA File	5 KB	



# JSP Lifecycle

- JSP Lifecycle (Contd)
- Following are the various phases present in lifecycle of a JSP page.



# JSP Scripting Elements

- JSP Scripting elements helps to embed java code in a jsp page.
- The scripting elements present in JSP along with syntax to include in JSP page are specified below.

Scripting Element	Syntax	Example
Declaration	<code>&lt;%! %&gt;</code>	<code>&lt;%! String track; %&gt;</code>
Scriptlet	<code>&lt;% %&gt;</code>	<code>&lt;% track = "Digital"; %&gt;</code>
Expression	<code>&lt;%= %&gt;</code>	<code>&lt;%= track %&gt;</code>
Comments	<code>&lt;%-- --%&gt;</code>	<code>&lt;%-- JSP comment --%&gt;</code>
Directive	<code>&lt;%@ %&gt;</code>	<code>&lt;%@page import="java.sql.Connection" %&gt;</code> <code>&lt;%@Taglib uri="/WEB-INF/mytag.tld" %&gt;</code> <code>&lt;%@include file="index.html" %&gt;</code>

# JSP Scripting elements - Declaration

- Declaration `<%! %>`
- Declaration tag contains the declarations of variables and methods required for processing the request.
- The statements embedded in this tag will be placed as instance variable and method of the generated servlet.
- The code can be placed inside `<%! %>` Element in JSP page

# JSP Scripting elements - Expression

- Expression    `<%= %>`
- This tag is used to print values to the web page.
- The statement specified here will be passed as argument to the `print()` method of the servlet.
- The code can be placed inside `<%= %>` Element in JSP page

# JSP Scripting elements - comments

- Comments    `<%-- --%>`
- Comments can be included in JSP page to provide description about the process.
- Comments are generally ignored by the compiler.
- The comment can be placed inside `<%-- --%>` Element in JSP page

# JSP Scripting elements - Scriptlet

- Scriptlet `<% %>`
- A scriptlet tag contains java statements that needs to be executed one after the other.
- The statements can be a variable declaration, initialization, expressions, method call, Object instantiation etc.
- The statement embedded in this tag will be placed as statements in the service method of the servlet.
- The code can be placed inside `<% %>` Element in JSP page

# JSP Scripting elements - Directive

- Directives `<%@ %>`
- In the translation phase JSP page is translated to a servlet.
- Directives are used to provide instruction to the web container on how to process this translation to generate the corresponding servlet.
- Following are the three types of directives that can be included in a JSP
  - Page directive
  - Include directive
  - Taglib directive
- Directives are associated with attributes which helps in JSP processing.
- The directives can be represented as
  - `<%@ directivename attributename="value" %>`

# JSP Scripting elements

- JSP Scripting Elements Example

Sample.jsp

```
<%@ page language="java" import="java.util.List"
contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://ww
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<%-- Declaration --%>
<%! String track; %>
<%-- Scriptlet --%>
<% track = "Digital"; %>
<%-- Expression --%>
<%= track %>
</body>
</html>
```



# JSP Scripting elements

- @Page directive
- The page directive is used to specify the attributes related to the entire JSP page.
- Following are the attributes which can be given in @Page directive

AttributeName	Description
Import	To import classes,interface,enums during the translation time
contentType	To set the response content type. Default value is text/html;charset=ISO-8859-1
pageEncoding	To set the response encoding type. Default value is ISO-8859-1
Extends	To specify the superclass of the generated servlet info
Buffer	To set the buffer size of the JSP page. Default value is 8kb.
Language	To denote the scripting language used in JSP page. Default value is java
isELIgnored	To give instruction to the container whether to conside EL or ignore it. Default value is false
isThreadSafe	To enable multithreading feature. Default value is false

# JSP Scripting elements

- @Page directive
- The page directive is used to specify the attributes related to the entire JSP page.
- Following are the attributes which can be given in @Page directive

AttributeName	Description
errorPage	To redirect request to a error page when exception occur in the JSP page.
isErrorPage	To denote whether the current JSP page can handle the exception and is set as error page.Default value is false
autoFlush	To flush the buffer automatically when it is full, Default value is true
Session	to retrieve the session values this can set to true.

# JSP Scripting elements

- @Include Directive
- This directive is used to include the content of a jsp/html page into the current JSP page.
- The contents are included before transition phase and mostly recommended for static content inclusions
- To include a page the following syntax can be used
  - `<%@include file="header.html" %>`

# JSP Scripting elements

## @Taglib directive

- This directive is used to include the tag library descriptor locations in order to use many predefined JSP tags.

```
<%@Taglib prefix="c" uri="http://java/jstl/jsp/coreTags" %>
```

# JSP implicit Objects

- Implicit Objects
- Implicit Objects are available in JSP to help the developer to use them directly in the page rather than creating them newly.
- There are nine implicit objects available in JSP.
- Following table describes the implicit objects precisely.

# JSP implicit Objects

- Implicit Objects

Object	Class or Interface	Description
out	PrintWriter in servlet is JspWriter in JSP.	Object of JspWriter class. This object is used mainly to produce output in the response webpage
request	HttpServletRequest in Servlet is request in JSP	Object of HttpServletRequest. This represents request object to get all the request from the user
response	HttpServletResponse in Servlet is response in JSP	Object of HttpServletResponse. This provides user response data from application to the client
config	ServletConfig in Servlet is config in JSP	Object of ServletConfig. This is used to hold the properties associated with the current jsp page
application	ServletContext in servlet is application in JSP	Object of ServletContext. This object holds the ServletContext parameters

# JSP implicit Objects

- Implicit Objects (Contd.)

Object	Class or Interface	Description
page	Object in Servlet is page in JSP	Object of Object class. This object indicates the current instance and helps to access the methods using the current instance
pageContext	javax.servlet.jsp.PageContext in servlet is pageContext in JSP	Object of PageContext. To set the scope of the page as request/response/session/application
exception	Exception in Servlet is exception in JSP	Object of Throwable class. This object is used to handle the exception that are redirected to the errorPage
session	HttpSession in Servlet is session in JSP	Object of HttpSession. This object is used to hold the session data maintained throughout the application

# JSP implicit Objects

- pageContext in JSP
- pageContext is an implicit Object in JSP page which is of type PageContext class.
- It is mainly used to set and get attributes using one of the following scopes
  - Page Scope: PAGE\_CONTEXT
  - Request Scope: REQUEST\_CONTEXT
  - Session Scope: SESSION\_CONTEXT
  - Application Scope: APPLICATION\_CONTEXT
- Page scope is default for this Object.

```
<%
```

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

```
out.print("Hello "+username);
```

```
pageContext.setAttribute("username",username,PageContext.SESSION_SCOPE);
```

```
<a href="nextpage.jsp">Go to next page</a>
```

```
%>
```

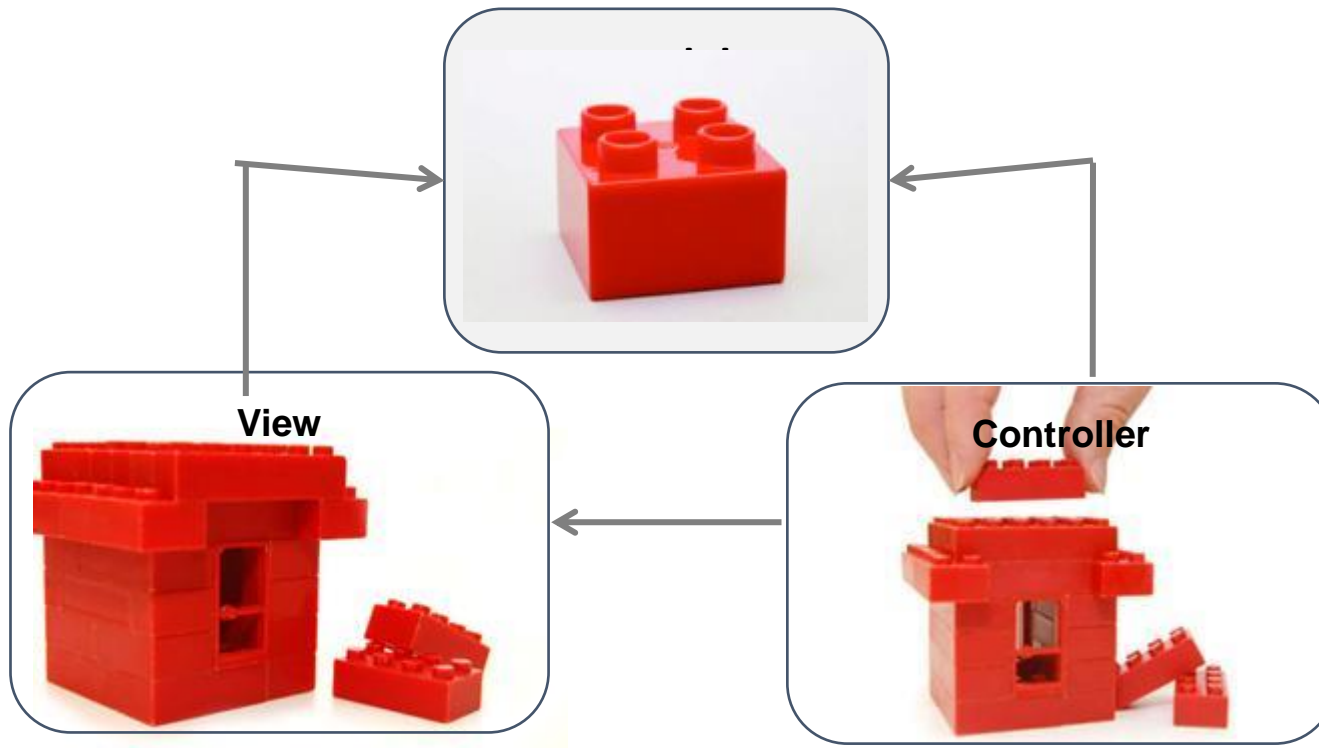


# Expression Language

- JSP EL
- Expression language in JSP page is used to access data that are stored in Java bean component in the application.
- These are used to include arithmetic as well as logical expressions in JSP page.
- Syntax used to denote Expression Language is `${ }`
  - `${ value>20 }`
  - `${ a + b }`
  - `${ param["LoginId"] }`

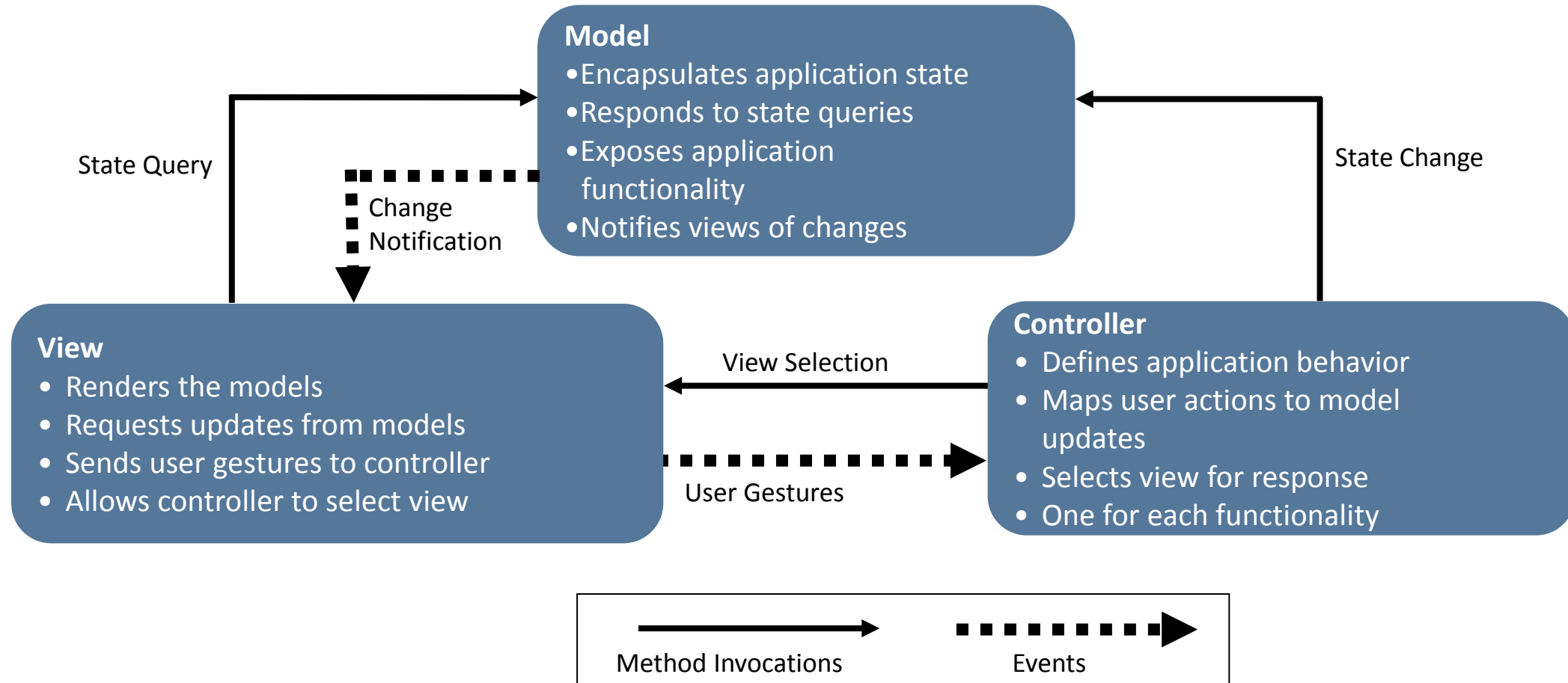
# Model-View-Controller (MVC)

- Components
- Structural relationship between the three components:



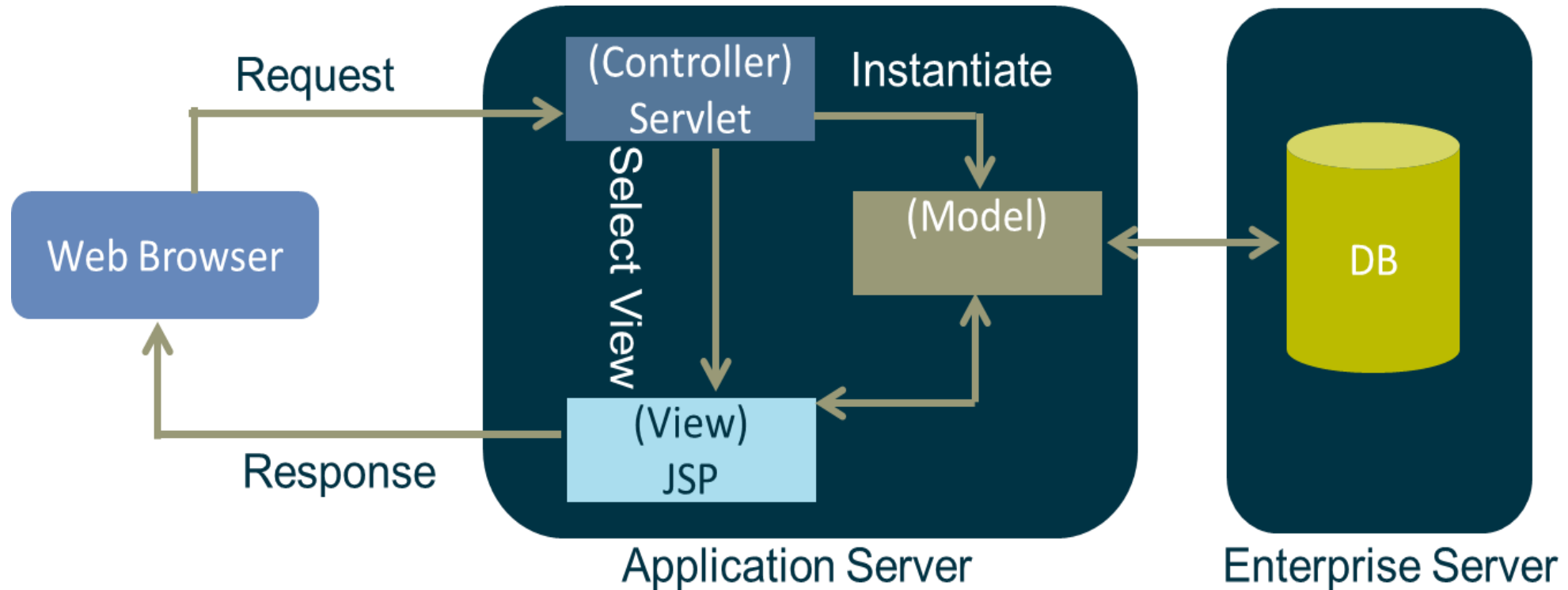
# Model-View-Controller (MVC)

- Components and its relationship



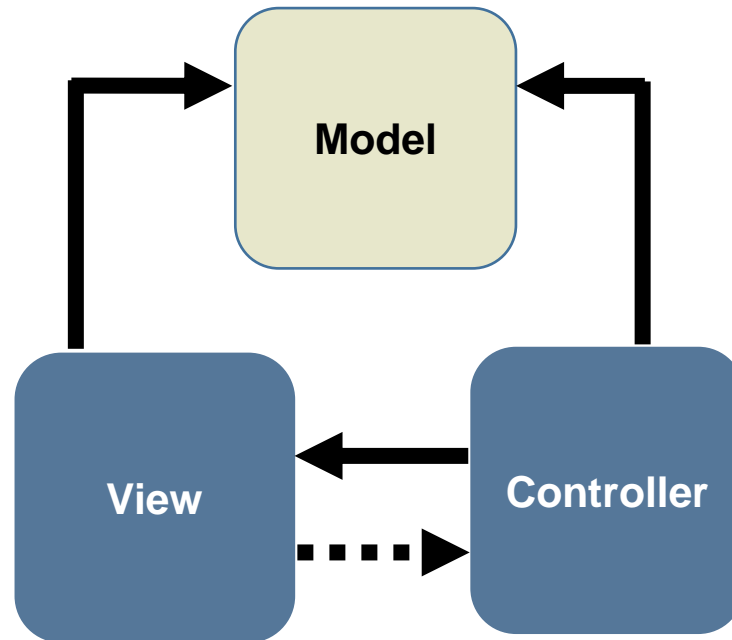
# Model-View-Controller (MVC)

- Model 2 Architecture



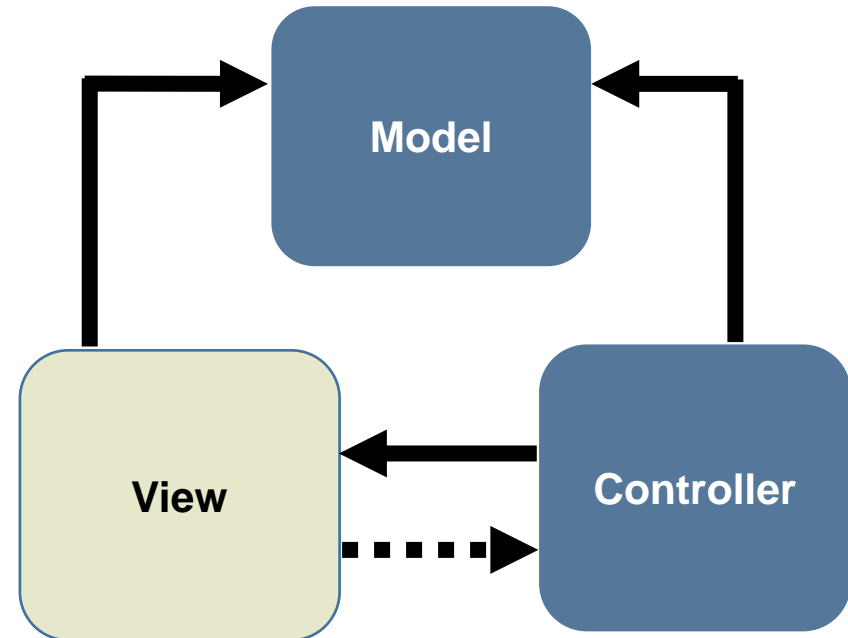
# Model-View-Controller (MVC)

- Model
- Knows about the data that needs to be displayed.
- Knows about the operations that can be applied on the data.
- Does not know how to display the data to the user.



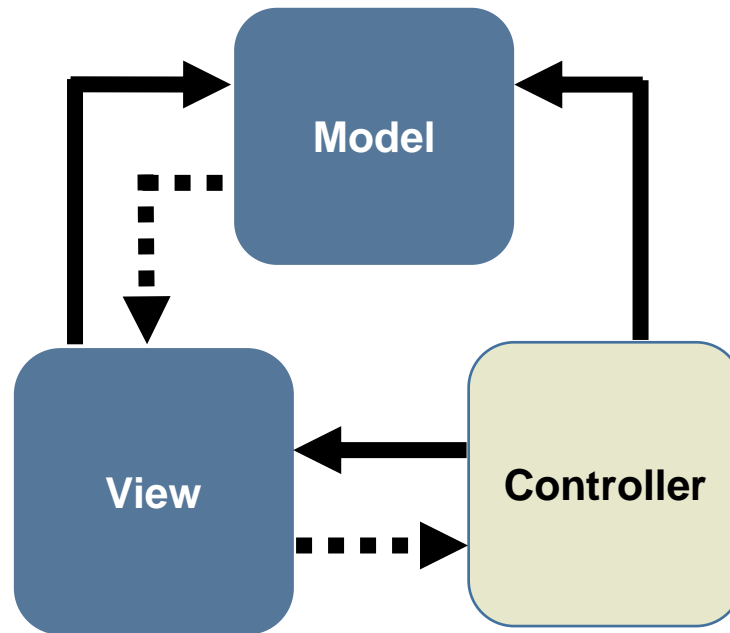
# Model-View-Controller (MVC)

- View
  - Provides Graphical User Interface components
  - Relays user requests
  - Uses the query methods
  - Displays information
  - Maintains consistency



# Model-View-Controller (MVC)

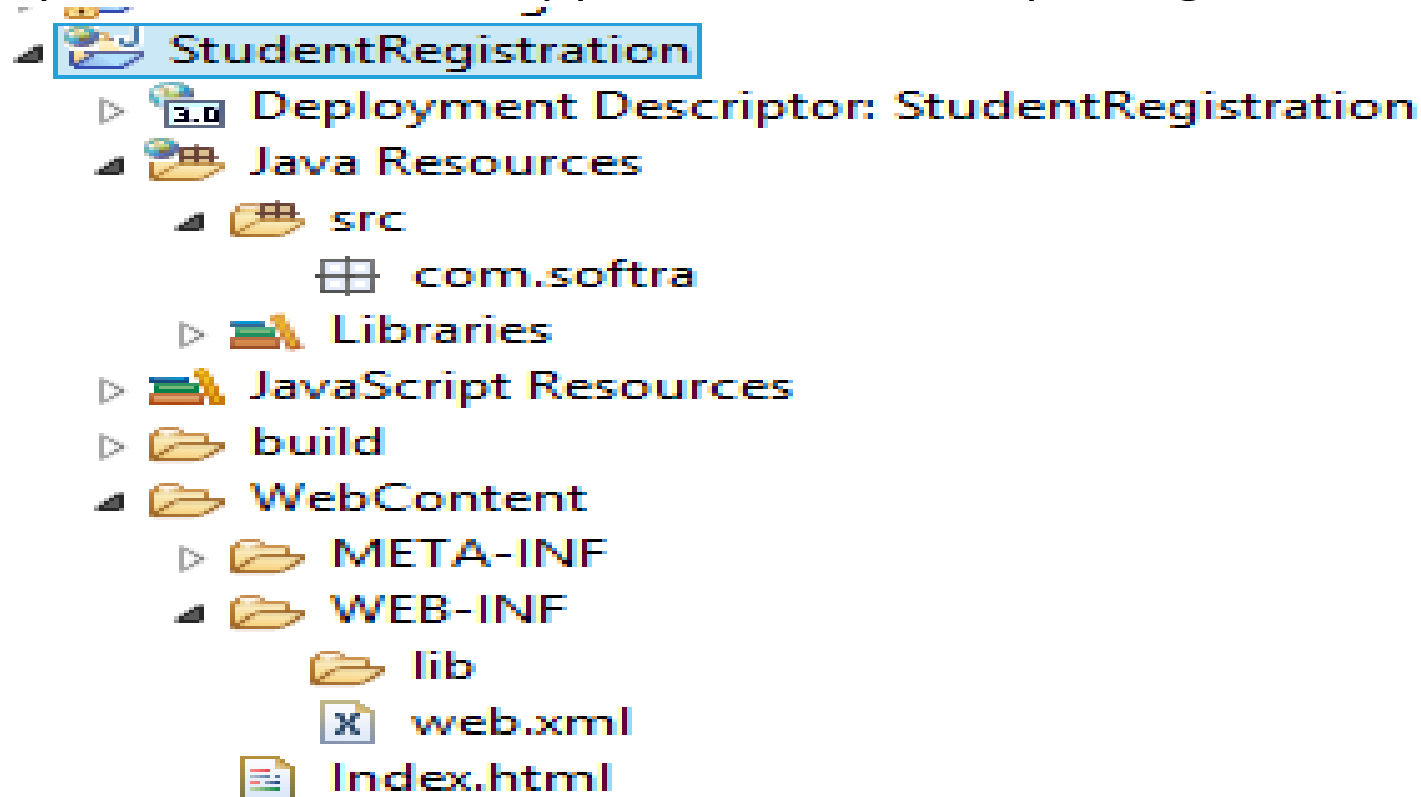
- Controller
- Translates interactions with the view into actions.
- Is responsible for mapping end-user action to application response.



# MVC : Case Study

- Activity - Create a web application (MVC pattern) to accept and display the student information.

Step 1: Create a web Application and the packages as follows:

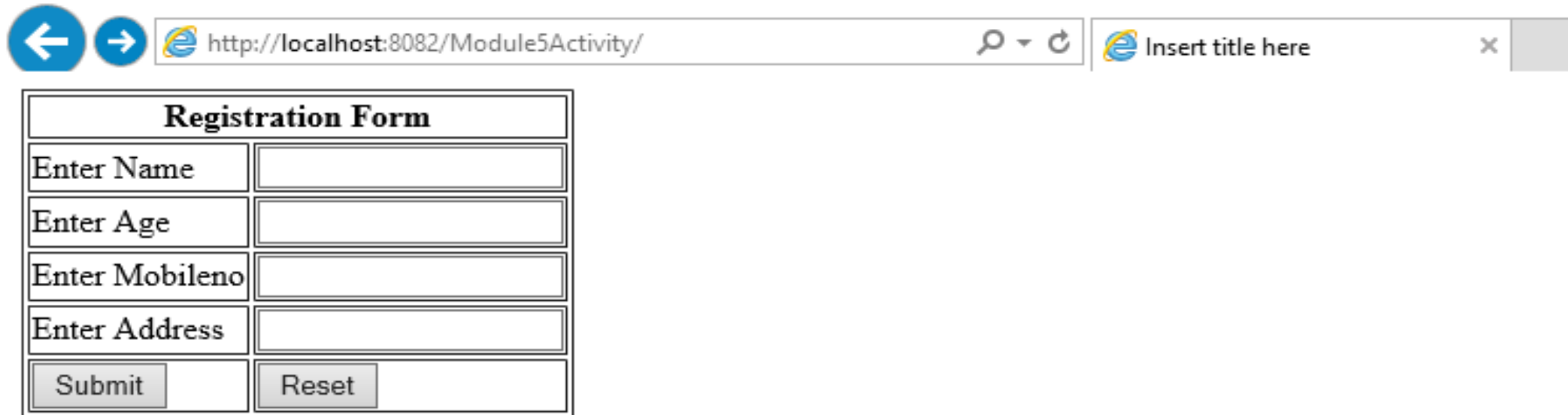




# MVC : Try It

- Activity - Create a web application (MVC pattern) to accept and display the student information.

Step 2: Use index.html to display the Student Registration form as follows:



The screenshot shows a web browser window with the address bar displaying `http://localhost:8082/Module5Activity/`. The page content is a registration form titled "Registration Form". The form contains four input fields for "Enter Name", "Enter Age", "Enter Mobilenos", and "Enter Address". At the bottom, there are two buttons: "Submit" and "Reset".

Registration Form	
Enter Name	<input type="text"/>
Enter Age	<input type="text"/>
Enter Mobilenos	<input type="text"/>
Enter Address	<input type="text"/>
<input type="button" value="Submit"/>	<input type="button" value="Reset"/>

# MVC : Try It

- Activity - Create a web application (MVC pattern) to accept and display the student information.

Step 3: Create a java class Student in the package com.softra.model (declare variables and getter setters)

Step 4: Create a servlet named RegisterServlet inside the package com.softra.controller(read the index.html form parameters, assign the values to the Student object and forward the request along with student object to the jsp page)

Step 5: Create ShowDetails.jsp to display the student information as follows:



Registered Successfully!!!

Your details are Name is : John

Age is : 23

Mobilenos : 9877663434

Address is : Mumbai

# Summary

- Create web application using Servlets and JSP
- Understand the lifecycle of Servlets and JSP
- Using ServletConfig and ServletContext Objects.
- Make use of RequestDispatcher
- Make use of Scripting elements and implicit objects in JSP
- Understand MVC Design Pattern

# Thank You