Servlets

Unit - 3

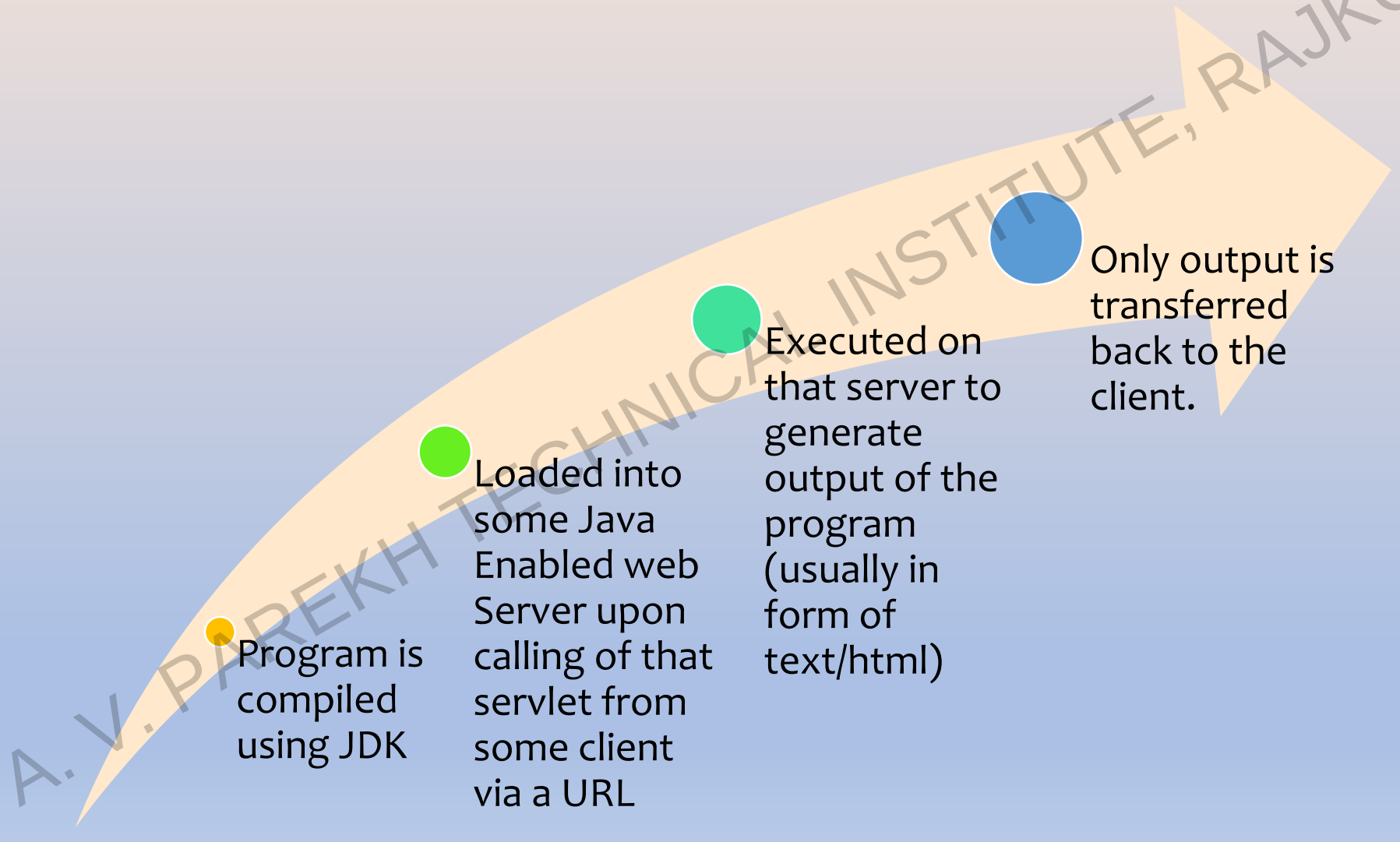
Topics to be Covered...

- Introduction to Servlets
- Life Cycle of Servlet
- Creating, Configuring & Deploying echo Servlet on Apache Tomcat
- Parameters and Attributes
  - HttpServletRequest(I)
  - HttpServletResponse (I)
  - ServletContext(I)
  - ServletConfig (I)
  - Request Delegation – RequestDispatcher (I)
- Session Tracking
- Database Connectivity in Servlet

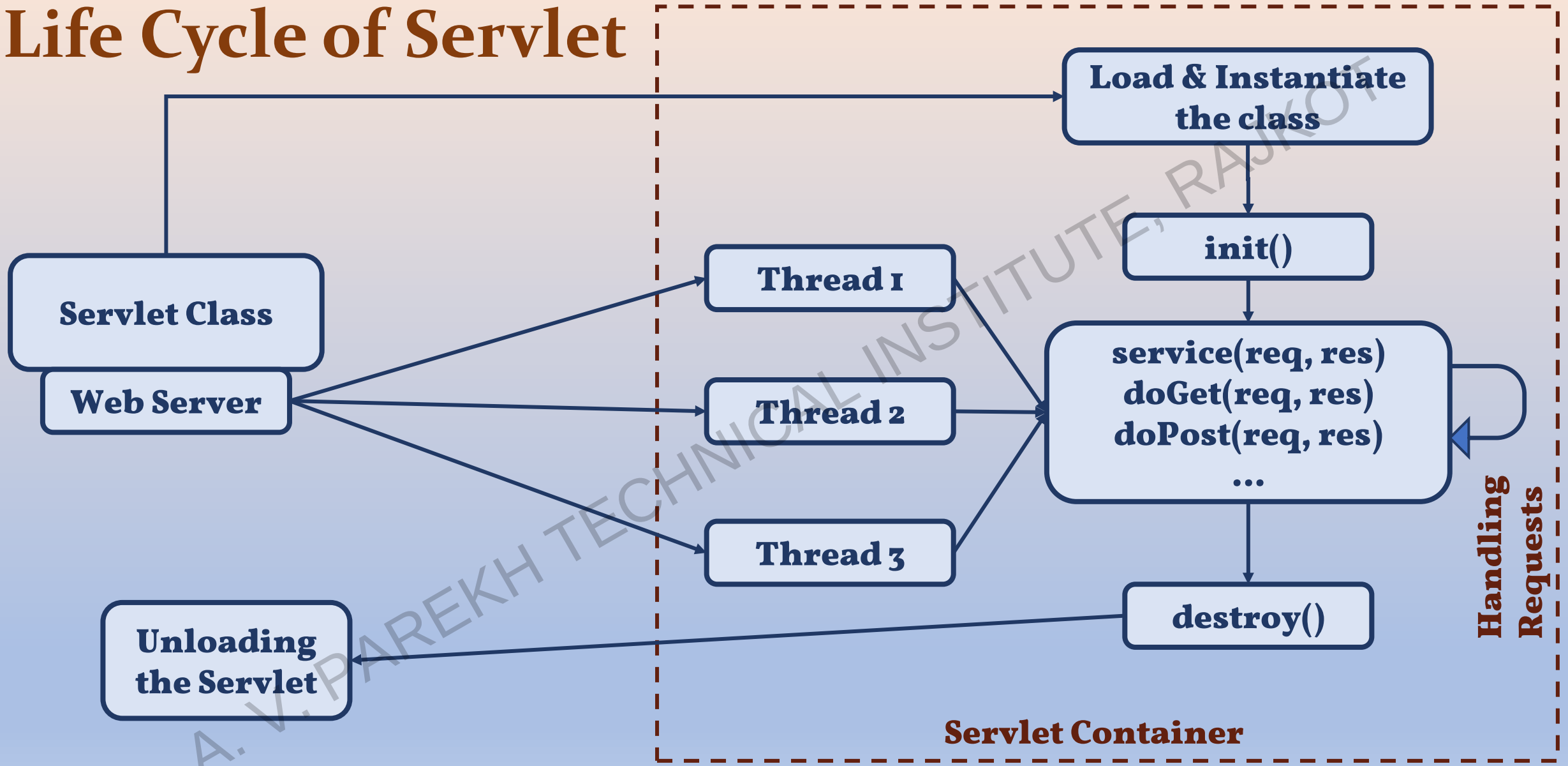
Introduction to Servlets

- Servlet is a sever extension of Java Program.
- An improved version of the CGI (Common Gateway Interfacae)
  - An older technology, popular back in the day for web programming.
  - Main Drawbacks were: Performance, Scalability, Reusability and Unable to handle multiple user request in a single session.
- It is a very popular technology due to its features like,
  - Platform Independence
  - Browser Independence
  - Code Reusability
  - Security
  - Performance
  - Server Extension
  - Extensibility

Servlets (Working Structure)



Life Cycle of Servlet



Life Cycle of Servlet

- A User requests a servlet using a URL from client browser via a URL.
- That URL is resolved and web server finds that servlet.
- If not loaded then, firstly the class of servlet is loaded into web container.
- After loading, instantiation is done by the container to make the servlet active.
- Initialization is done next to initialize some parameters needed to be initialized before making the servlet available to users. **init()** method is called to perform user defined initialization tasks.

```
public void init(ServletConfig config) throws ServletException
// can be overridden to perform specific tasks.
```

- Called only once per life cycle, very first method to be called, just after initialization of the servlet class.

Life Cycle of Servlet

- After the initialization is complete, the web container starts executing the next methods for servlet, which is capable of handling user requests and give responses in return.
- service() / doGet() / doPost() is executed at that time.
- For GenericServlet:

```
public void service(ServletRequest request, ServletResponse
                response) throws ServletException, IOException;
```

- For HttpServlet

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

- doGet can be replaced with any method of HTTP Protocol like doPost, doHead, doRequest, doDelete etc. with the same signature. We can override any number of methods that we need.
- This method can be executed for any number of user requests.

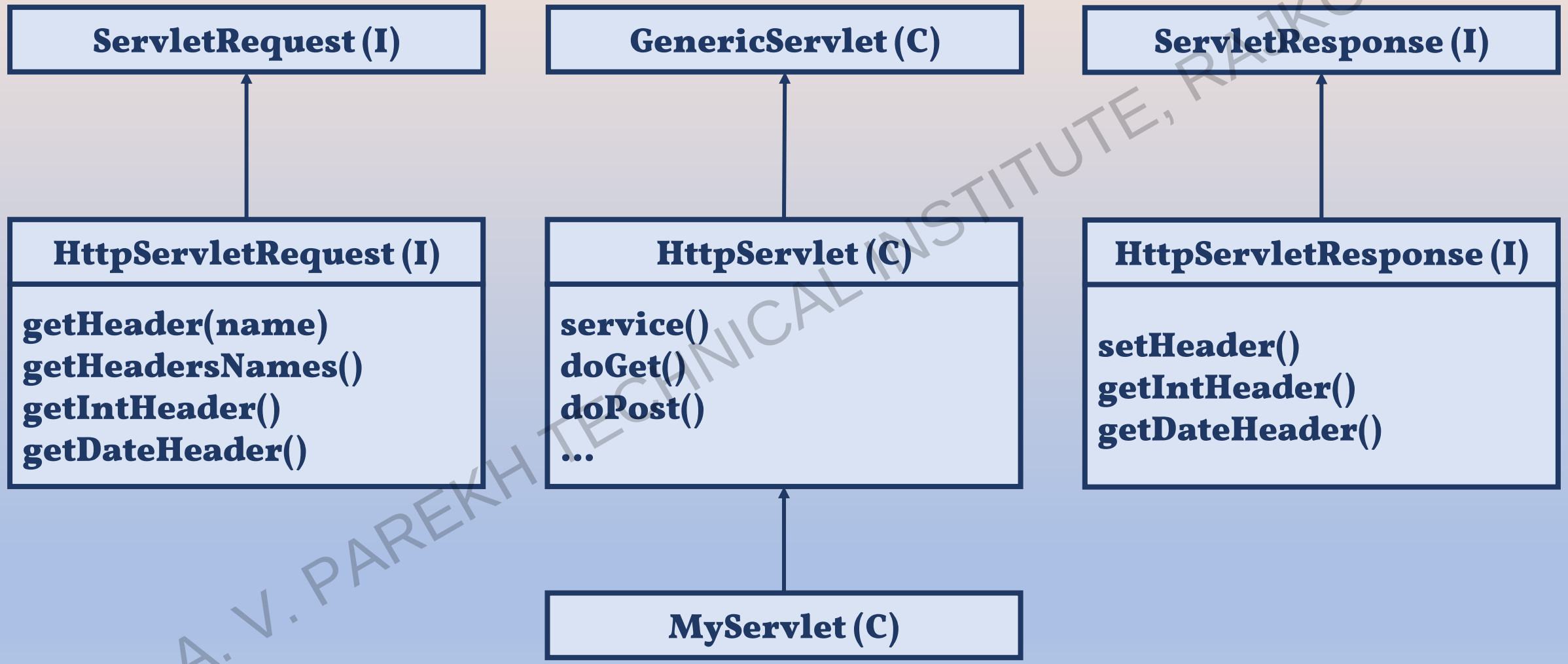
Life Cycle of Servlet

- After all the requests are handled, when the user closes the servlet window, the destroy() method is called.
- Called only once per life cycle, in the last, just before destroying the servlet object.
- Signature:

```
public void destroy();
```

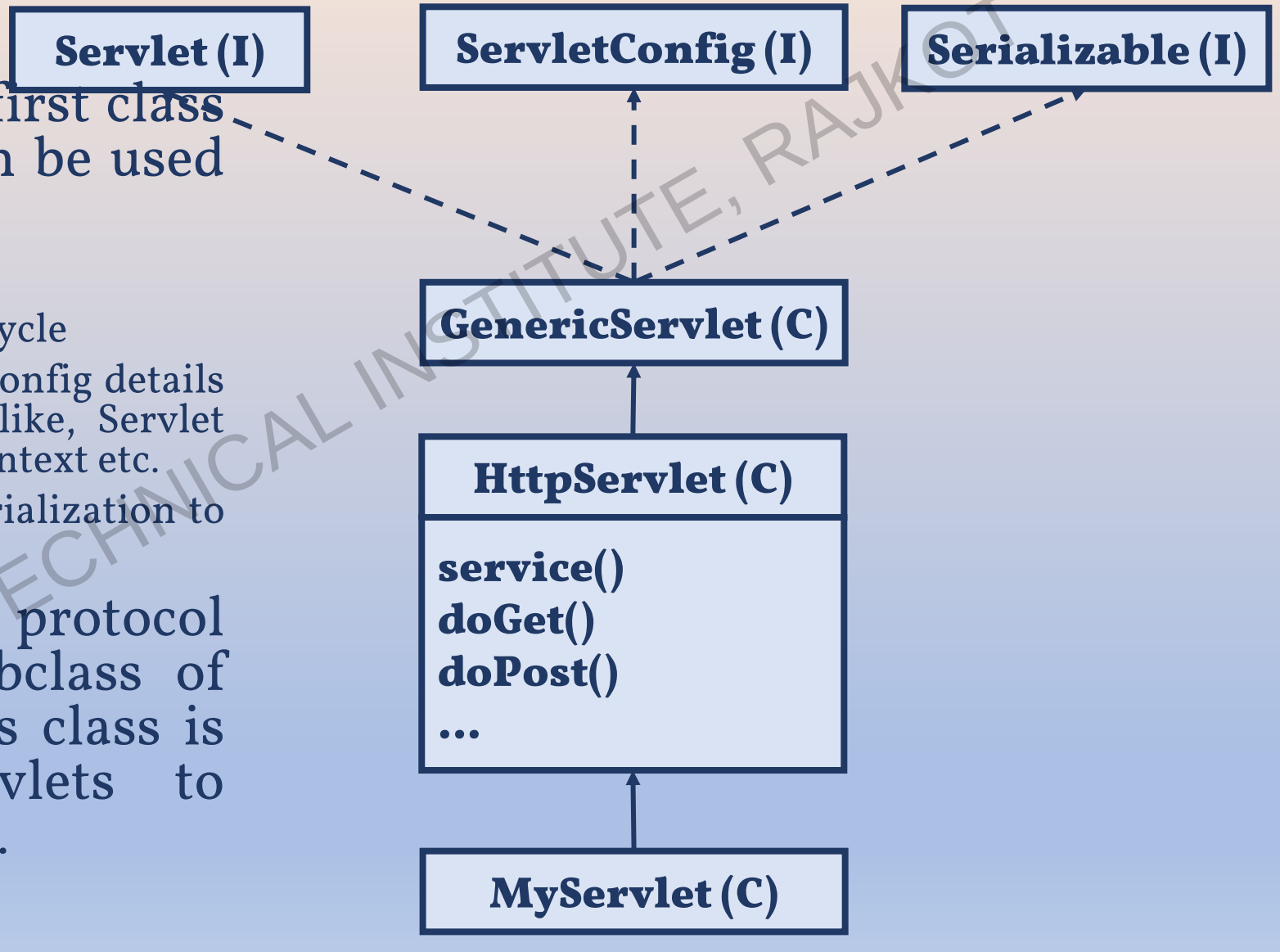
- Method usually contains code to free up space or closing some open connections.
- After the destroy method is executed, the web container will destroy the servlet object and Unload the class eventually to free up the space.

Servlet API



Servlet Class

- GenericServlet class is the first class in the hierarchy, which can be used to develop servlets.
  - It is derived from 3 interface:
    - Servlet (I): To provide its life Cycle
    - ServletConfig (I): To provide config details while executing the Servlet like, Servlet Name, Init Params, Servlet Context etc.
    - Serializable (I): To provide serialization to the servlets.
- HttpServlet, an HTTP protocol specific servlet, is the subclass of GenericServlet, usually this class is inherited to create servlets to execute over HTTP protocol.



Servlet Methods

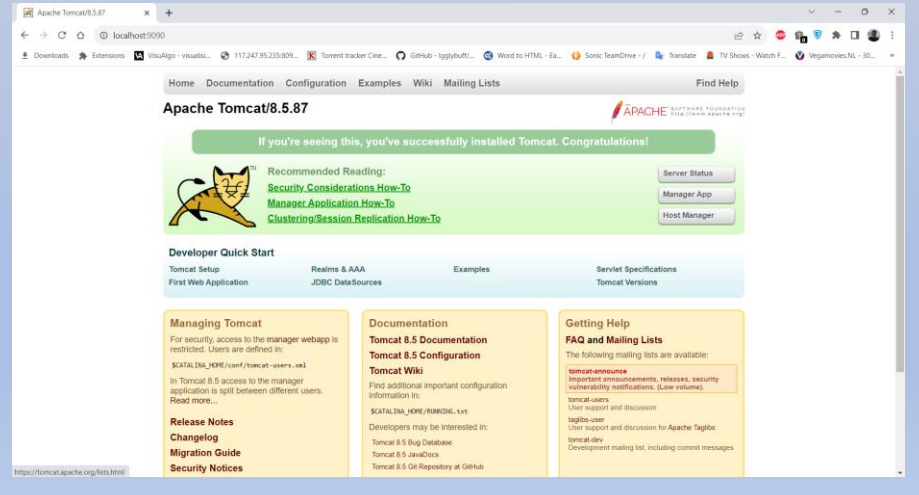
Method	Description
`public void init(ServletConfig config) throws ServletException`	It is called when the Servlet is first initialized or created. Web container calls the init method once after initializing the Servlet, UnavailableException is thrown if Servlet is initialized.
`public ServletConfig getSetvletConfig()`	Returns ServetConfig object containing current configuration of Servlet.
`public void service(ServletRequest reqest, ServletResponse response) throws ServletException, IOException`	This is the main method called by web container while servicing servlet for each user requests. request and response are the objects that handles user requests and give response to users. If anything happens to IO then IOException is thrown and if anything happens to Servlet then ServletException is thrown.
`public String getServletInfo()`	This method returns Servlet information about current Servlet.
`public void destroy()`	This method is only called once when the servlet is being destroyed from web container.

Servlet Configuration: Installing Tomcat

- To execute servlet firstly a Server is needed which is Java Enabled
  - Apache Tomcat (Free & Open Source)
  - Glassfish
  - JBoss
- Steps to install & Run Apache Tomcat:
  - Setup JDK on your computer.
  - Download the setup (Windows Service Installer) of Apache Tomcat from: https://tomcat.apache.org/download-10.cgi (Any suitable version can be downloaded)
  - Start the set up and change necessary attributes like Log In credentials, Connector Port etc.
    - Changing the port to 9090 from 8080 is recommended, to avoid clash with other services (like Oracle) running on the same port.

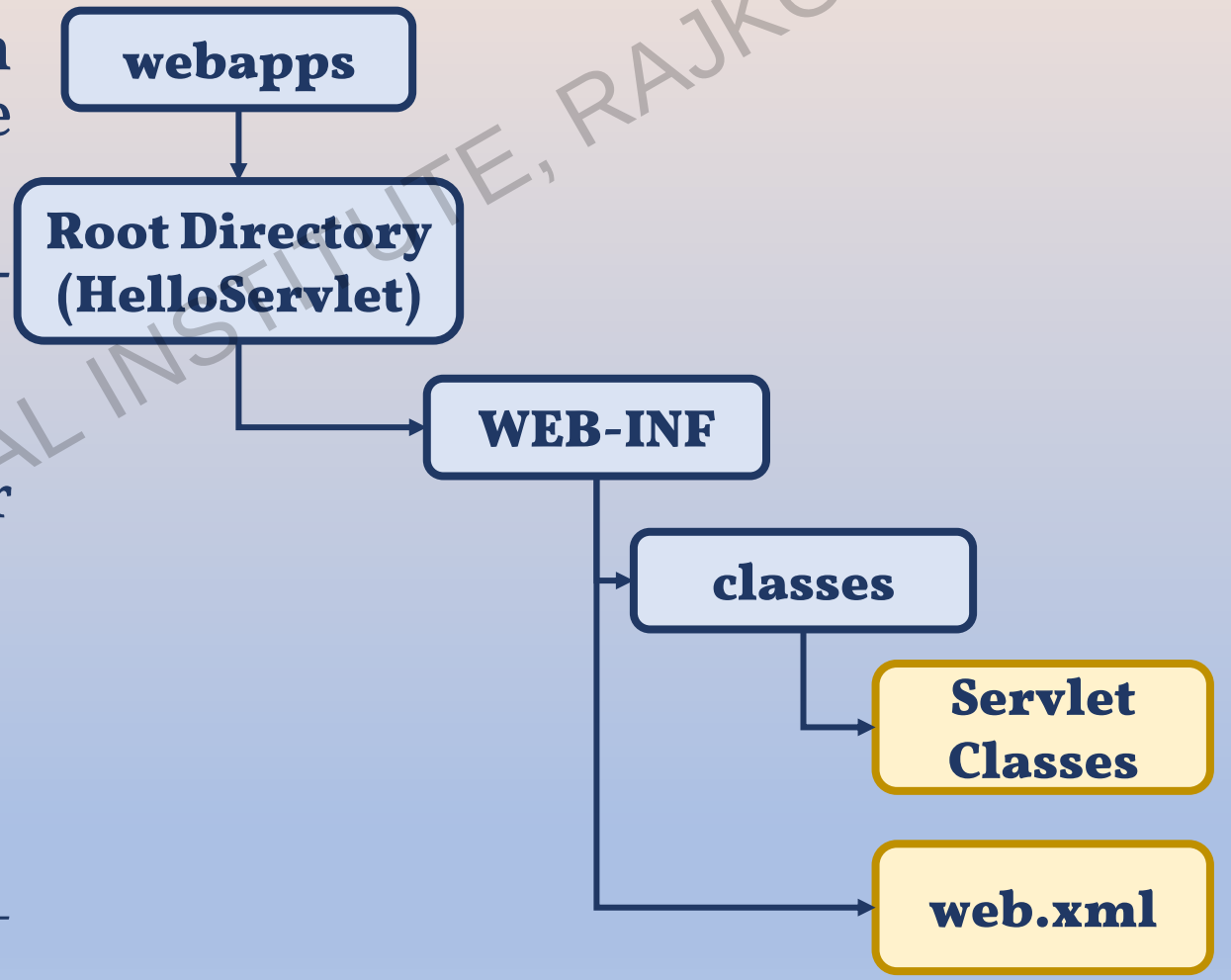
Servlet Configuration: Installing Tomcat

- After the set-up, the Environment Variable “classpath” needs to be set to the path of servlet-api.jar file, available at “<tomcat-installation-directory>/tomcat<version>/lib/servlet-api.jar”.
- After setting this, Java compiler can find Servlet API classes for compilation and execution of the servlet.
- To check installation, Just Start Tomcat server and type URL: localhost:9090



Servlet Configuration: Creating Directory Structure

- Servlets are supposed to execute in Server, so it needs to be put in the server container directory
- Which is "<tomcat-installation-directory>/tomcat<version>/webapps" by default.
- Inside that create a root directory for you website.
- In that directory, create 2 things
  - "WEB-INF" directory
    - Create a directory "classes"
      - Put servlet class file here.
  - web.xml file (Deployment Descriptor File – Used to map URL <-> Servlet class)



Servlet Configuration: Creating Servlet Class

```
//import necessary classes
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.GenericServlet;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.ServletException;

// Create a class of type GenericServlet
public class HelloWorldServlet extends GenericServlet {
```

Servlet Configuration: Creating Servlet Class

```
    //Override necessary methods
    public void service (ServletRequest request,
ServletResponse response) throws IOException, ServletException {
        //Set Output Content Type
        response.setContentType("text/html");
        //Create an out object to write in the webpage
        PrintWriter out = response.getWriter();
        //Write your Code.
        out.print("<H1>Hello World from Servlet!!!");
        //Close the out object
        out.close();
    }
} //Class Termination
```

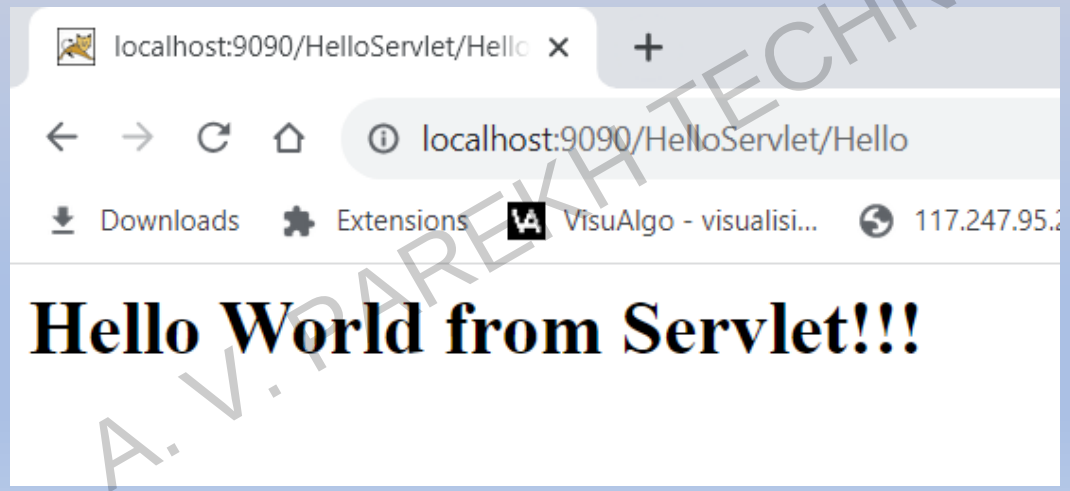
Servlet Configuration: Creating Deployment Descriptor

- web.xml file is known as the deployment descriptor file.
- File contains the description of all components of the web.

```
<?xml version="1.0" encoding="utf-8"?>
<web-app> <---Root element
      <servlet> <---To map Class name and Servlet name
            <servlet-name>FirstServlet</servlet-name>
            <servlet-class>HelloWorldServlet</servlet-class>
      </servlet>
      <servlet-mapping> <--To map Servlet name to a URL
            <servlet-name>FirstServlet</servlet-name>
            <url-pattern>/Hello</url-pattern>
      </servlet-mapping>
</web-app>
```

Servlet Configuration: Compiling & Execution

- To Execute the servlet, first compile it.
  `…/classes>javac HelloServlet.java`
- This should be compiled without any errors.
- Now to execute it, just start the Tomcat server and type its url
  `localhost:9090/HelloServlet/Hello`



HttpServlet

- HttpServlet is a protocol specific servlet where service() method is replaced with HTTP protocol specific actions.
- Other changes required are objects of request, response are replaced with protocol specific classes: HttpServletRequest & HttpServletResponse
- Methods instead of service are:
  - doGet()
  - doPost()
  - doHead()
  - doPut()
  - doDelete()
  - doOptions()
  - doTrace()
- Any number of methods can be overridden in a single servlet.
- The appropriate methods will be chosen at the time of execution as per the protocol method used.

ServletConfig

- Servlet can access Init parameters passed from web.xml file.
- Usually it contains servlet wide common constant data.

```
<servlet>
    <init-param>
        <param-name>dbName</param-name>
        <param-value>StudentDB</param-value>
    </init-param>
    <servlet-name>FirstServlet</servlet-name>
    <servlet-class>HelloWorldServlet</servlet-class>
</servlet>
```

ServletConfig

- Those can be accessed using ServletConfig object from servlet.

Method	Description
`public String getInitParameter(String param)`	Returns a String containing the value of the specified initialization parameters or null if the parameter does not exist.
`public Enumeration getInitParameterNames()`	Returns the names of all initialization parameters as an Enumeration of String objects. If no initialization parameters have been defined, an empty Enumeration is returned.
`public ServletContext getServletContext()`	Returns the ServletContext object for the servlet, which allows interaction with the Web container.

```
//Code in servlet
ServletConfig config = getServletConfig();
String dbName = config.getInitParameter("dbName");
```

ServletContext

- Some Init parameters can be application wide as well.
- Like some port number to access some device/data
- That can be provided through web.xml file as well.

```
<web-app>
    <context-param>
        <param-name>portNo</param-name>
        <param-value>9090</param-value>
    </context-param>
</web-app>
```

ServletContext

```
//Code in servlet
ServletCotext context = getServletContext();
String portNo = context.getInitParameter("portNo");
```

ServletConfig v/s ServletContext

ServletConfig	ServletContext
ServletConfig object is one per servlet	ServletContext object is global to the entire web application.
This object can be created during the initialization of the servlet	This object is created at the time of web application deployment
This object is public to the particular servlet only and is destroyed as soon as the servlet execution is completed.	This object will be persistent and available to the whole application. This can be destroyed when the whole application is repmoved from the servlet
We need to request server explicitly, in order to create ServletConfig object for the first time.	ServletContext object will be always available for servlet to use even before giving the first request.
<init-param> tag will appear under <servlet> tag in web.xml	<context-param> tag will appear under <web-app> tag in web.xml
getServletConfig() is used to get this object	getServletContext() is used to get this object
The interface has set of other methods like, getInitParameter(String name), getInitParameterNames(), getServletName(), getServletContext()	The interface has set of other methods like, getInitParameter(String name), getServletName(), getServletContext()

Cookie in Servlet

- Some temporary data can be stored and accessed by the servlet from browser.
- This is a client side storage.
- Cookies are small text files that are stored by an application server in the client browser to keep track of all the users.
- A cookie has values in the form of name/value pairs.
- It can be sent from Servlet to browser with Response object headers.
- A Web browser is expected to support 20 cookies per host and the size of each cookies can be a maximum of 4 bytes each.

Cookie in Servlet

- Characteristics of Cookies are:
  - Cookies can only be read by the application server that had written them in the client browser.
  - Cookies can be used by the server to find out the computer name, IP address or any other details of the client computer by retrieving the remote host address of the client where the cookies are stored.
- Types of Cookies:
  - **Session Cookies:** Session cookies do not have expiration time. It lives in the browser memory. As soon as the web browser is closed this cookie gets destroyed.
  - **Persistent Cookies:** Unlike Session cookies they have expiration time, they are stored in the user hard drive and gets destroyed based on the expiry time.

Cookie in Servlet

- Cookie can be created using following steps:
  - Create a Cookie object.

```
Cookie c = new Cookie("userName","john");
```

  - Set the maximum Age: By using setMaxAge () method we can set the maximum age for the particular cookie in seconds.

```
c.setMaxAge(1800);
```

  - Place the Cookie in HTTP response header: We can send the cookie to the client browser through response.addCookie() method

```
response.addCookie(c);
```

Cookie in Servlet

- Reading cookies back from browser to Server.
- Cookies can be get From the Request object headers.

```
Cookie c[] = request.getCookies();
//c.length gives the cookie count
for(int i = 0; I < c.length; i++){
    out.print("Name: "+c[i].getName()+" & Value"+c[i].getValue());
}
```

Cookie in Servlet

Method	Description
`Cookie(String name, String value)`	Contructor to create a new cookie object with specified name and value
`void setMaxAge(int expiry)`	Sets Expiry Time is seconds
`Void getName()`	Gets the name of the cookie
`void getValue()`	Gets the value of the cookie
`void setValue(String value)`	Sets the value of the cookie

`void addCookie(Cookie c)` //Method of HttpServletResponse to add cookie object to the browser

`Cookie[] getCookies()` //Method of HttpServletRequest to get array of cookies from the browser

Session in Servlet

- Session management is the process of keeping track of the activities of a user across Web pages.
- Consider an example of an online shopping mall.
  - The user can choose a product and add it to the shopping cart.
  - When the user moves to a different page , the details in the shopping cart are still retained ,so that the user can check the items in the shopping cart and then place the order.

Session Management

- HTTP is a stateless protocol and therefore cannot store the information about the user activities across Web pages.
- However , there are certain techniques that helps store the user information across Web pages using HTTP protocol.
- The techniques that you can use to maintain the session information are:
  - Hidden form field
  - URL rewriting
  - Cookies
  - Servlet session API

Hidden Form Field

- You can use hidden form fields to maintain the session information of a user while the user interacts with the web application.
- A hidden form field is embedded in an HTML page and is not visible when viewed in a browser.
- The following code snippet shows a hidden form field in an HTML page:

```
<html>
<form method="get" action=" ">
<input type="hidden" name="id" value="L234">
-------------
---------------
</html>
```

URL Rewriting

- URL rewriting is a session management techniques that manages user session by modifying a URL .
- Usually ,this techniques is used when information that is to be transferred is not very critical because the URL can be intercepted easily during transfer.
- For example , in an online shopping portal, servlet can modify a URL to include user information, such as a username.

```
http://localhost:9090/DataServlet?val1=20&val2=40
```

Session API of Servlet

- The HttpSession object is used for session management.
- A session contains information specific to a particular user across the whole application.
- When a user enters into a website (or an online application) for the first time HttpSession is obtained via request.getSession(), the user is given a unique ID to identify his session.
- This unique ID can be stored into a cookie or in a request parameter.

Session API of Servlet

- Creating a session (if available returns the object, if not – creates new):

```
protected void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
  HttpSession session = req.getSession();
}
```

- Setting Session Attributes in the same servlet:

```
session.setAttribute("uName", "Jack");
session.setAttribute("uemailId", "jack@mail.com");
session.setAttribute("uAge", "48");
```

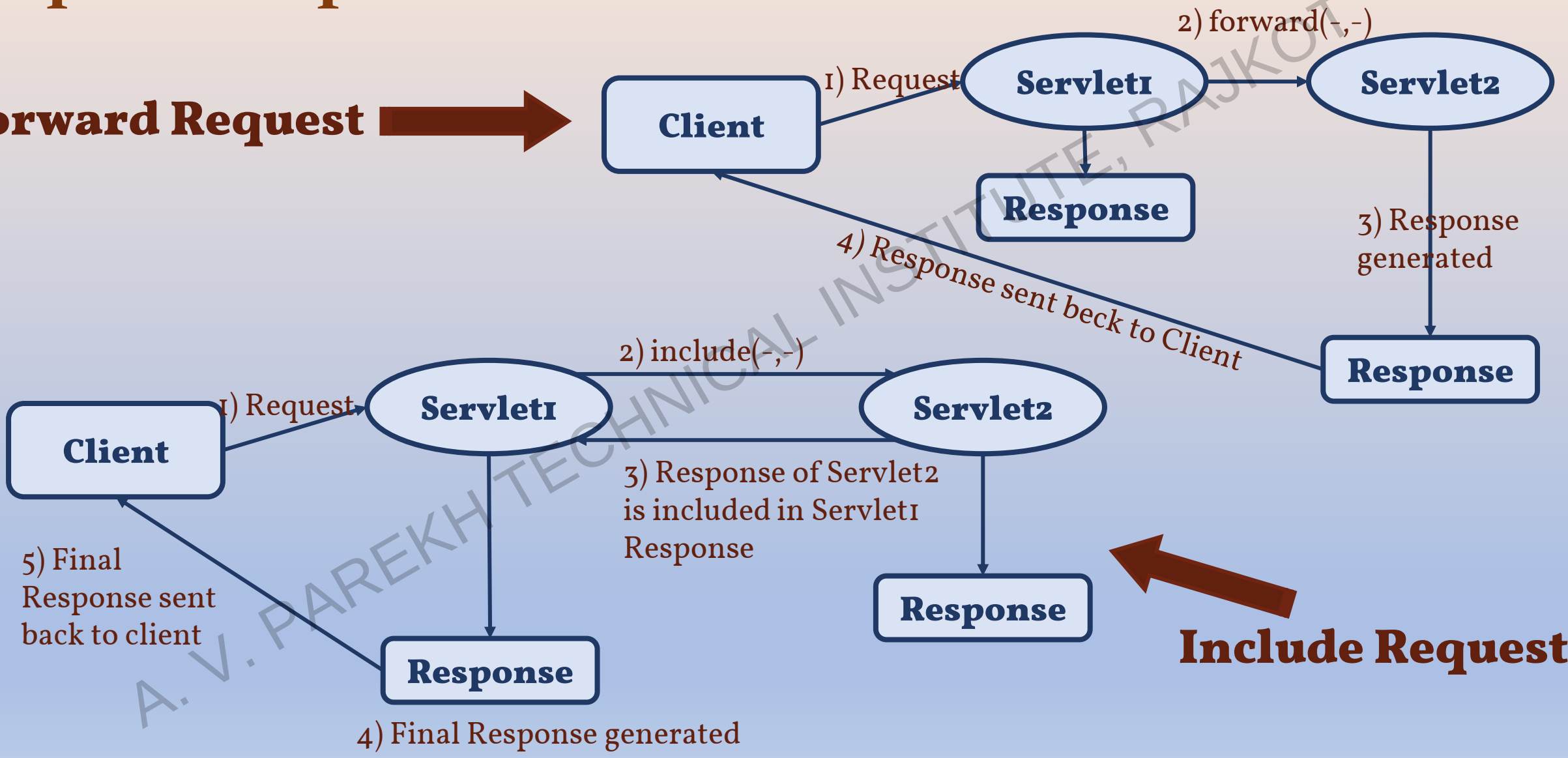
- Accessing Session from another page:

```
HttpSession session = request.getSession(false); //Don't create a new session
String userName = (String) session.getAttribute("uName");
String userEmailId = (String) session.getAttribute("uemailId");
String userAge = (String) session.getAttribute("uAge");
```

RequestDispatcher in Servlet

- The RequestDispatcher interface allows you to route the request to another resource, which could be html, servlet, or jsp.
- This interface can also be used to include the content of a different resource. It is a method of servlet collaboration.
- 2 Methods are:
  - public void forward(ServletRequest request, ServletResponse response) throws ServletException, IOException: Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
  - public void include(ServletRequest request, ServletResponse response) throws ServletException, IOException: Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

RequestDispatcher in Servlet



RequestDispatcher in Servlet

- Usage of Forward Request:

```
RequestDispatcher rd = request.getRequestDispatcher("servlet2");
rd.forward(request, response);
```

- Usage of Include Request

```
RequestDispatcher rd=request.getRequestDispatcher(“servlet2");
rd.include(request, response);
```

Another way to redirect in Servlet

- sendRedirect() method:
  - The sendRedirect() method of HttpServletResponse interface can be used to redirect response to another resource, it may be servlet, jsp or html file.
  - It accepts relative as well as absolute URL.
  - It works at client side because it uses the url bar of the browser to make another request. So, it can work inside and outside the server.
  - Signature:

```
public void sendRedirect(String URL) throws IOException;
```

  - Usage:

```
response.sendRedirect("http://www.google.com");
```

forward() v/s sendRedirect()

forward() method	sendRedirect() method
The forward() method works at server side.	The sendRedirect() method works at client side.
It sends the same request and response objects to another servlet.	It always sends a new request.
It can work within the server only.	It can be used within and outside the server.

Example:
`request.getRequestDispacher("servlet2")`
`.forward(request,response);`
	Example:
	`response.sendRedirect("servlet2");`