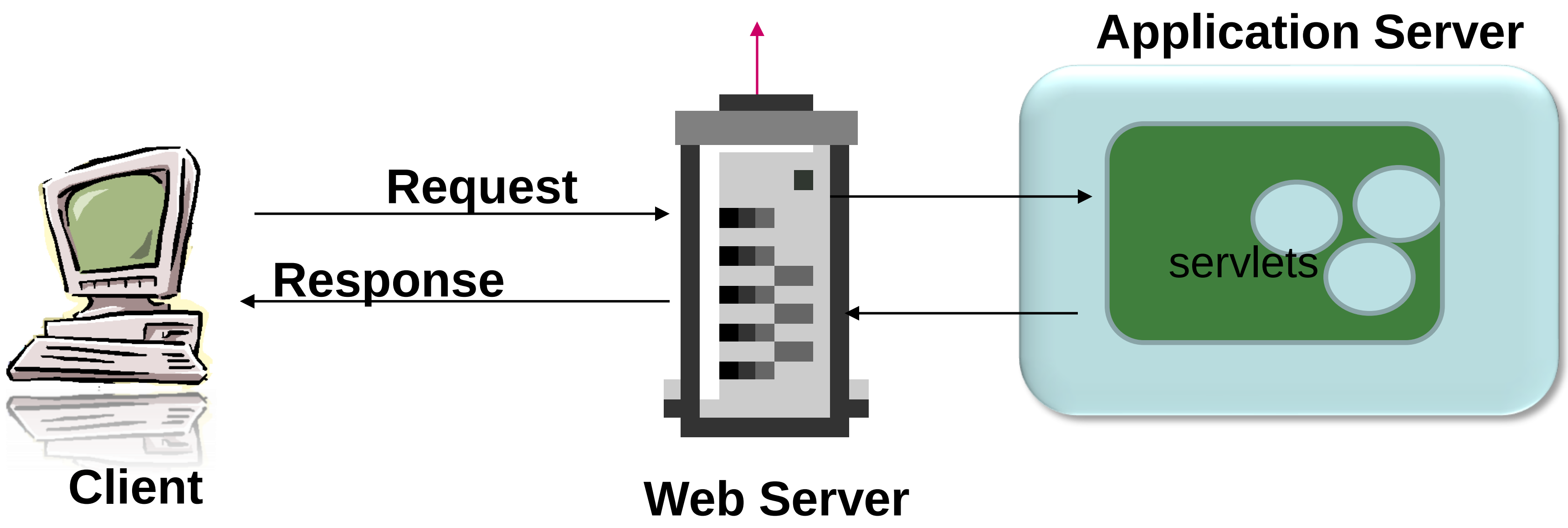
Servlet: Introduction

Definition

- A Servlet is a server side program written in Java that resides and executes in an JEE application server.
- It extends the functionality of a web server by enabling the delivery of dynamic content by fetching the data from various data sources and working on them.
- Servlets can respond to any type of request but they are commonly used to respond to HTTP/HTTPs request .
- JEE application server that hosts the Servlets is called Servlet Container or Servlet Engine.
- Servlet 3.0 specification is the one that will be discussed in the session. Underlying Java platform used will be Java SE 6.
- Also we will be using Servlets to respond to HTTP requests.

Typical servlet request-response mechanism

Webserver→If request is for a static page→ return the page as response else
Pass it to Application server →request is for Servlet→
Pass it on to Servlet container →locate the Servlet that has to be invoked



Here application server and web server is logically separated. They are physically in the same machine.

Servlet API

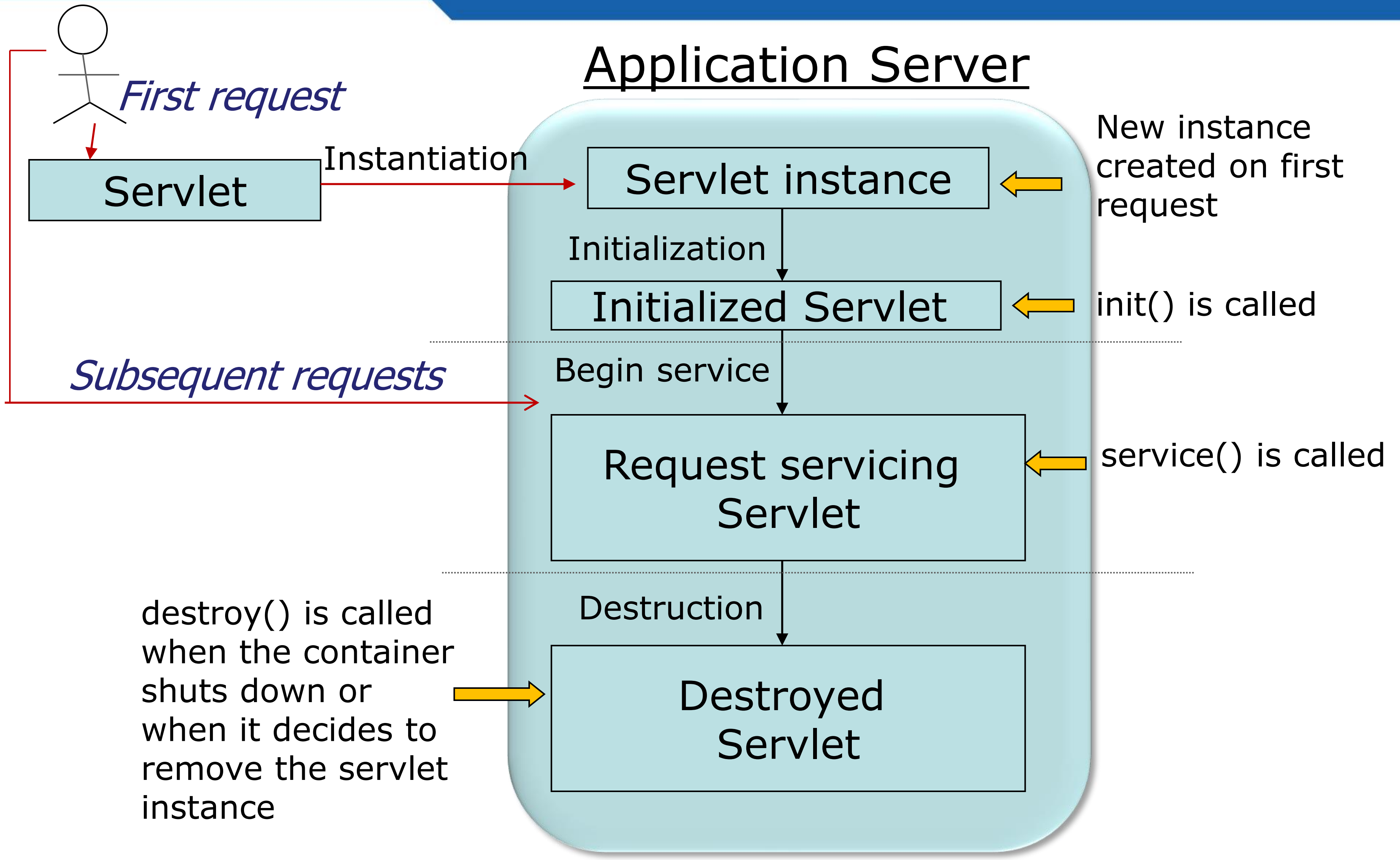
- Java Servlet API 3.0 is an API of the Java Platform, Enterprise Edition (JEE)
- Java Servlet Development Kit (JSDK) contains the class library that is required to create servlets and JSP.
- The 2 packages that are required to create servlets are **`javax.servlet`** and **`javax.servlet.http`**.
- The core of Servlet is **`javax.servlet.Servlet`** interface. A java class is a servlet if it directly or indirectly implements the **`javax.servlet.Servlet`** interface.
- **`Servlet`** interface has life cycle methods.
- Life cycle tells when the servlet be created and what happens during the life of the servlet and when does it get destroyed.

Servlet interface

Life cycle methods

- `public void init(ServletConfig config) throws ServletException`
- `public void service(ServletRequest req, ServletResponse res) throws ServletException, java.io.IOException`
- `public void destroy()`
- `public ServletConfig getServletConfig()`
- `public String getServletInfo()`

Servlet Life Cycle



Servlet Life Cycle - states

- Creation and Initialization:
  - Typically when the first request comes for a Servlet, if a Servlet instance does not exist, container
    - Loads the class and creates an instance of the Servlet
    - initializes the servlet instance by calling the **`init()`** method
  - The loading and instantiation of servlet can occur when the container is started as well. This is completely at the discretion of the container.
- Request Processing:
  - When request comes for a Servlet, if the instance exists, then service() is invoked and the request and response objects are passed.
  - Generally the concurrent requests to the same servlet is by executed by different threads invoking the **`service()`** of the same servlet instance
- Destroying Servlet
  - The container finalizes the servlet by calling the servlet’s destroy method before removing the servlet.

Servlet instance

- In an undistributed environment, only one instance of a servlet class is created. The same instance is used for all the requests.
- When the container shuts down, this instance is destroyed.
- The container runs multiple threads on the service() method to process multiple requests.
- Only a single instance of servlet of each type ensures minimum number of servlet objects created and destroyed on the server which in turn adds to scalability.
- However, if the Servlet implements **`SingleThreadModel`** interface, the servlet container may instantiate multiple instances. This is usually done to handle a heavy request loads. It is recommended that this is very sparingly and not used for the sake of thread-safety.
- In distributed environment, one servlet instance per JVM is created.

`init(ServletConfig config)`

- **`public void init(ServletConfig config) throws ServletException`**
- This method is called exactly once after instantiating the servlet.
- The init method must complete successfully before the servlet can receive any requests.
- If init() method throws a **`ServletException or UnavailableException`**, or if it does not return after certain time period defined by the application server, then the **`service()`** and **`destroy()`** is not called.
- **`ServletConfig`** object has initialization and startup parameters for the servlet. → *Will look into this a bit later*

`service()`

- **`public void service(ServletRequest req, ServletResponse res) throws ServletException, java.io.IOException`**
- This method is called after the **`init()`** method returns successfully.
- Since the **`service()`** method will be called by multiple threads, it is very important to synchronize access to any shared resources such as files, network connections, and also instance variables of the Servlet.
- It is strongly recommended not to synchronize the service() (or methods dispatched to it like doGet(), doPost → *coming up*) as it may effect on performance.
- This method may throw **`ServletException`** in case of some errors or **`UnavailableException`** in case servlet is unable to handle requests either temporarily or permanently.

`destroy()`

- **`public void destroy()`**
- This method is called once all threads within the servlet's **`service()`** method have exited or after a timeout period for the servlet is over.
- After this method is called, **`service()`** method will not be called again on this servlet.
- Usually clean up code like releasing resources etc is done put here.

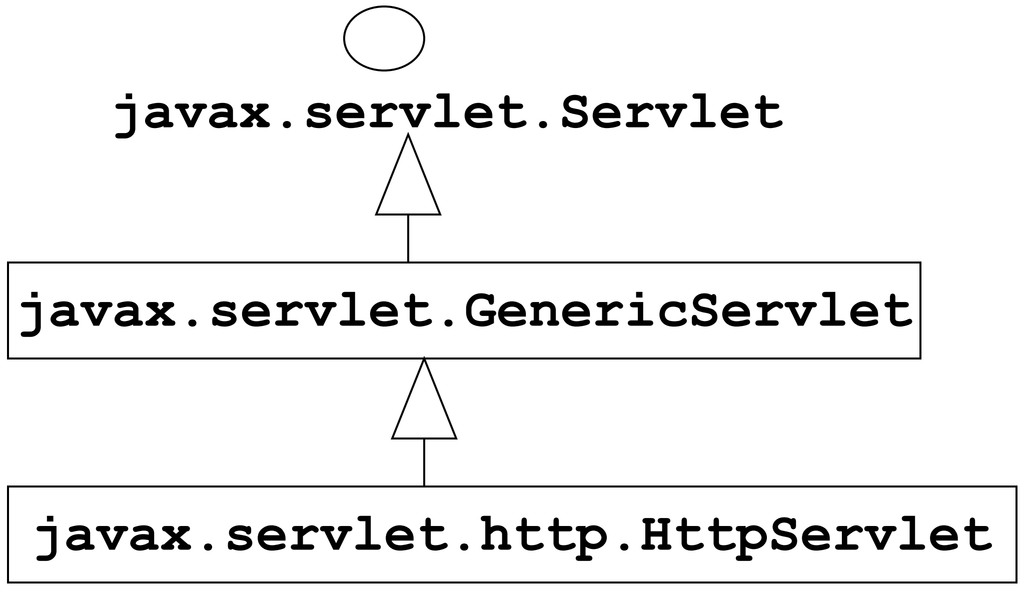
`getServletConfig(),getServletInfo()`

- **`public ServletConfig getServletConfig()`**
  - Returns **`ServletConfig`** object which is passed to init() method.
  - We will see how to use this object in the next session
- **`public String getServletInfo()`**
  - Can be used to return about the servlet, such as author, version, and copyright.
  - Usually the response string from the servlet is in the form of HTML.
  - But here, the response should not be in form of HTML or XML. It must be plain text.

Tell me why

- Why the `service()` method should throw `IOException`?
- A servlet returns the response by using a `Writer` object which it creates using response object.
- We are well aware that most of the methods of `Writer` object throw `IOException`.

`Servlet` Class hierarchy



GenericServlet

- **`GenericServlet`** class provides basic implementation of the **`Servlet`** interface except **`service()`** method. This is an abstract class.
- It also implements **`ServletConfig`** interface.
- Other methods included are:
  - **`public void log(String message)`**
  - **`public void log(String message, Throwable t)`**
  - **`public void init() throws ServletException`**
- Log methods are used to write specified message to a servlet log file.
- The implementation of **`init(ServletConfig config)`** calls **`init()`** method.
- For most purposes, we will need to extend **`HttpServlet.`**

HTTPServlet

- This is also an abstract class which inherits from the **`GenericServlet`** and provides a HTTP specific implementation of the **`Servlet`** interface.
- The **`service(ServletRequest req, ServletResponse res)`** method is implemented and handles standard HTTP requests by dispatching them to the handler methods for each HTTP request type .
- This class adds another service method that takes HTTP requests and response.
- Handler methods specific to each of the HTTP methods like GET, POST etc is also added in the form of doGet(), doPost() etc
- This is the class that we subclass and override one or more of the **`doXXX()`**.

Methods added in `HTTPServlet`

- `protected void service(HttpServletRequest req, HttpServletResponse res) throws ServletException, java.io.IOException`

This

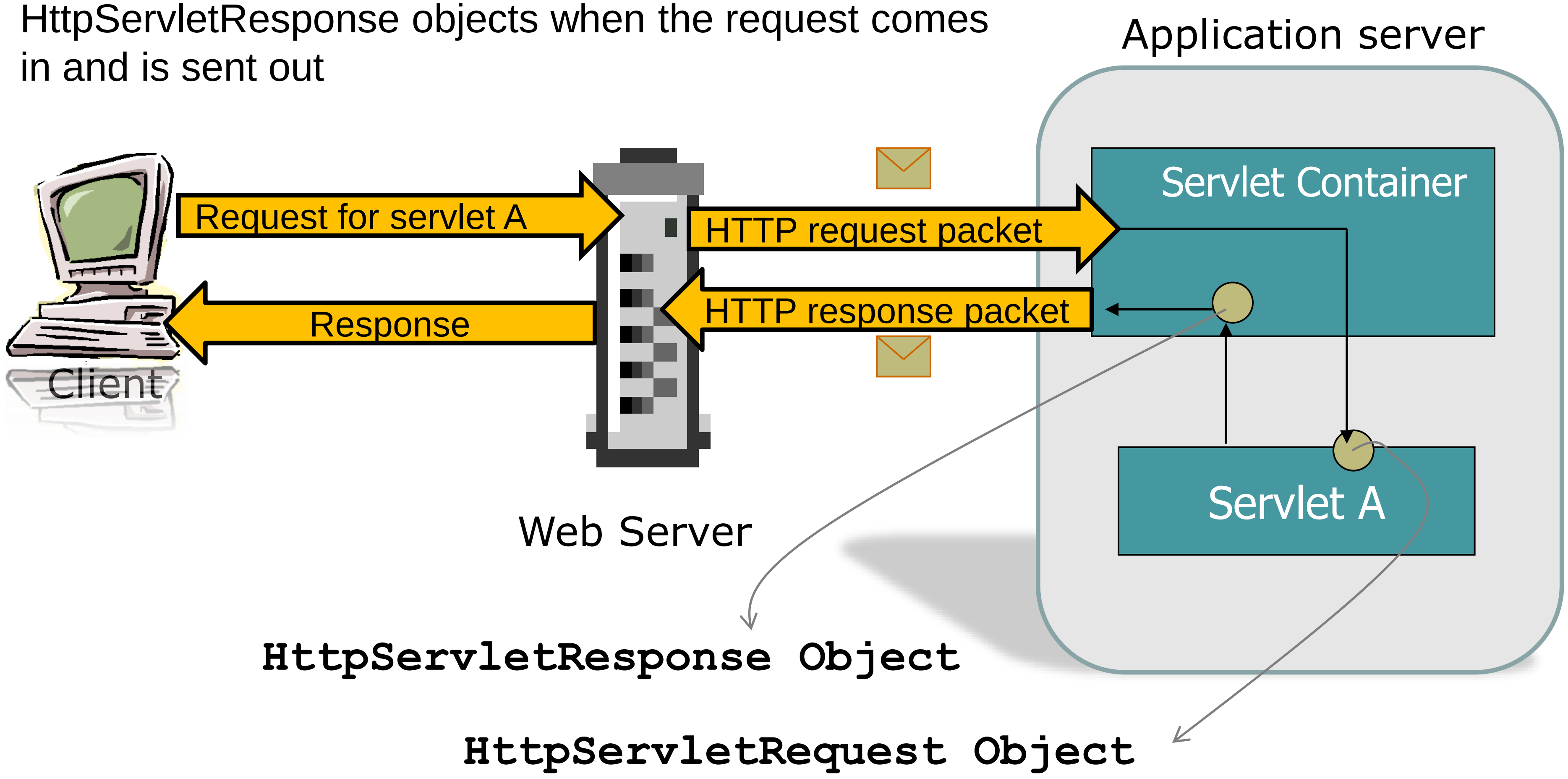
- `protected void doXXX(HttpServletRequest req, HttpServletResponse res)`

`XXX is GET, POST, TRACE, PUT, DELETE, HEAD, OPTIONS`

- `protected long getLastModified(HttpServletRequest req)`
- It is recommended that `service()`, `doOptions()` and `doTrace()` methods are not overridden.

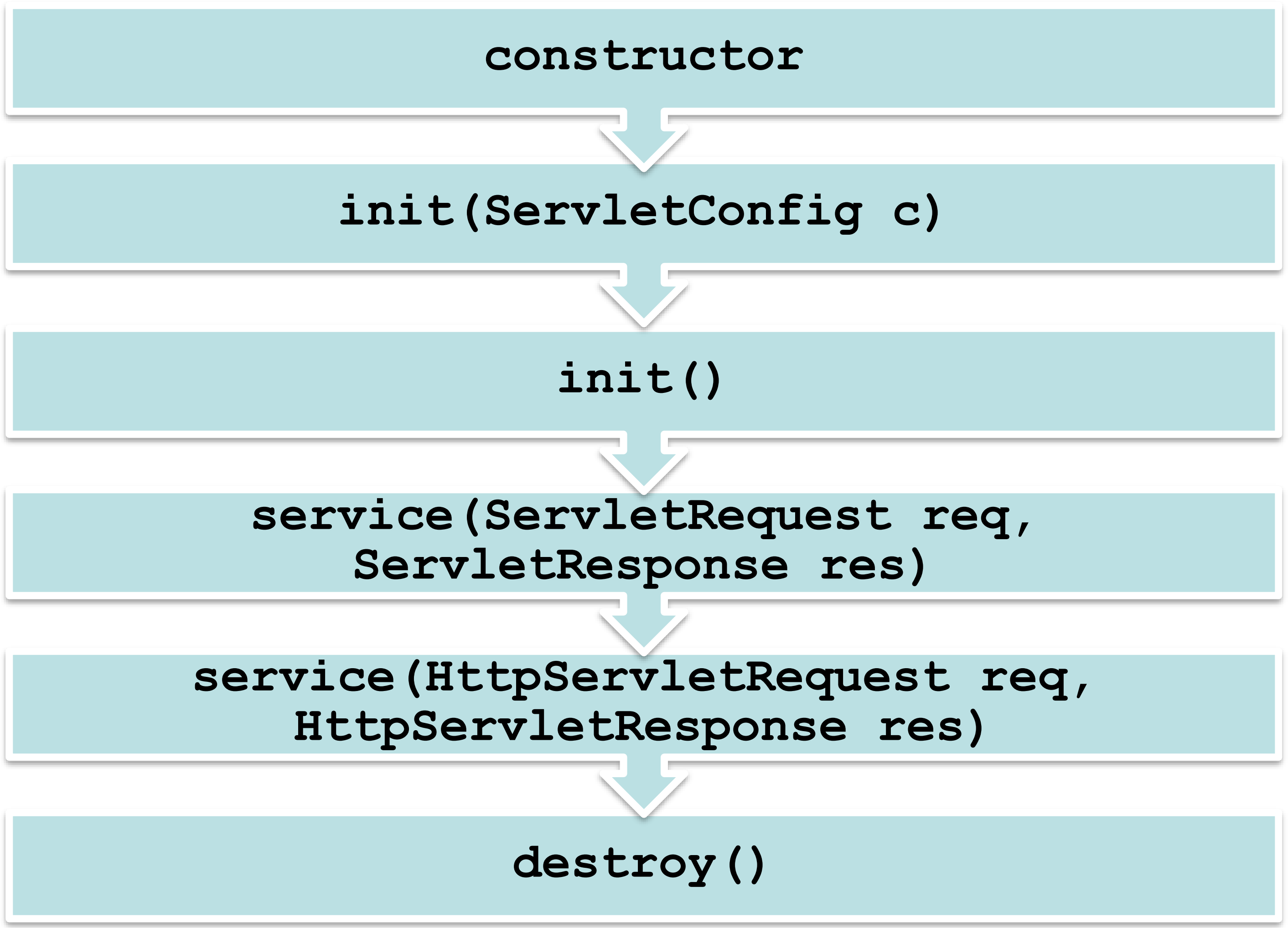
Request and Response object

Servlet container creates HttpServletRequest and HttpServletResponse objects when the request comes in and is sent out



The **`PrintWriter`** object is obtained from the **`HttpServletResponse`** object and the output is written

Lifecycle methods of a HttpServlet object


constructor
init(ServletConfig c)
init()
service(ServletRequest req, ServletResponse res)
service(HttpServletRequest req, HttpServletResponse res)
destroy()

A Simple Servlet

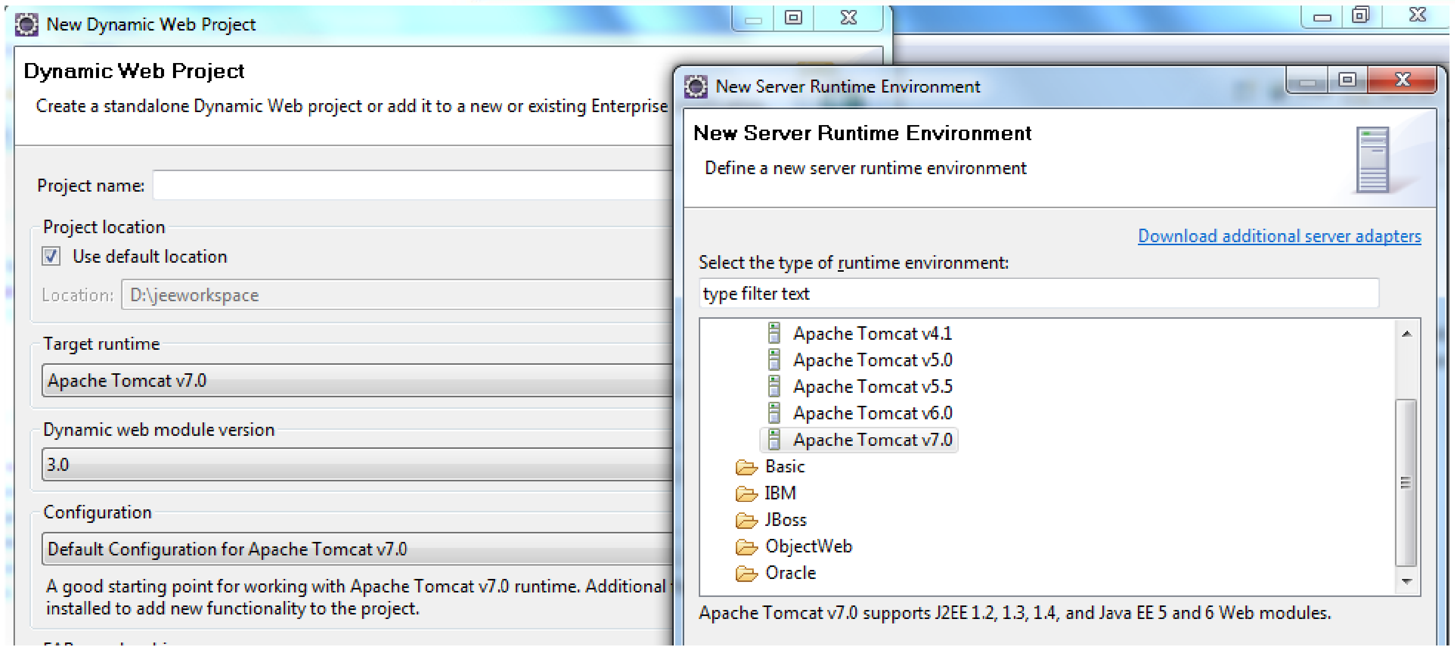
/* This example overrides `doGet()` method. It dynamically generates an HTML page that says Hello and prints current date-time of the server.*/

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class GreetingServlet extends HttpServlet {
public void doGet(HttpServletRequest request,
   HttpServletResponse response) throws ServletException,
   IOException {
  java.util.Date currDate=new java.util.Date();
  java.io.PrintWriter out = response.getWriter();
  response.setContentType("text/html");
  out.println("<html><head><title>Greetings Servlet
  </title></head>");
 out.println("<body>Hello "+currDate+"</body></html>");
}
```

Deploy a web application Eclipse Java EE IDE

1. Create web application.
2. Create servlets and override doGet()
3. Map Servlet through annotations or deployment descriptor (DD) web.xml.
4. Deploy and test the application

1. Create web application



New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise

Project name:

Project location

☑ Use default location

Location: D:\jeeworkspace

Target runtime

Apache Tomcat v7.0

Dynamic web module version

3.0

Configuration

Default Configuration for Apache Tomcat v7.0

A good starting point for working with Apache Tomcat v7.0 runtime. Additional installed to add new functionality to the project.

New Server Runtime Environment

New Server Runtime Environment

Define a new server runtime environment

Download additional server adapters

Select the type of runtime environment:

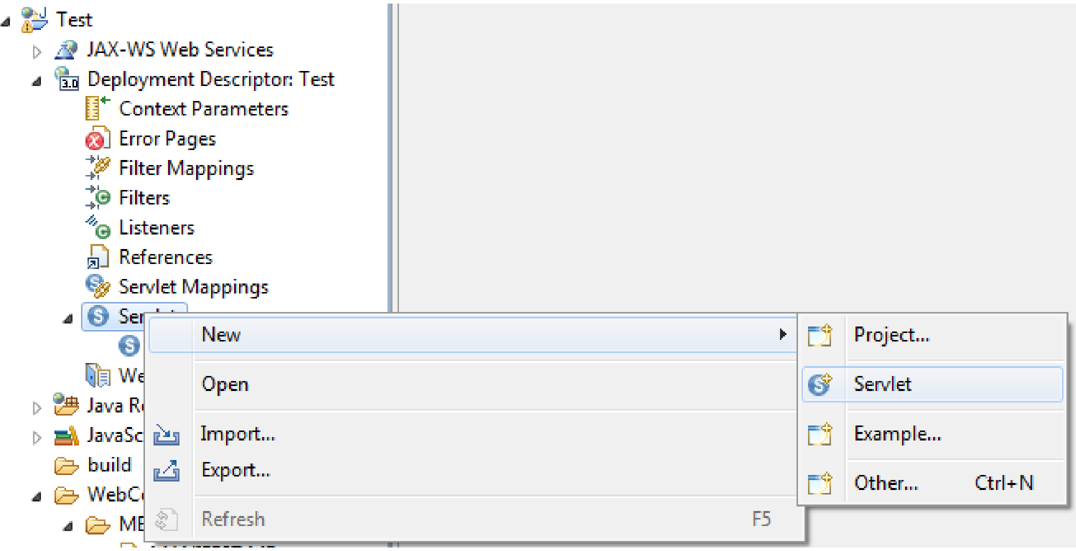
type filter text

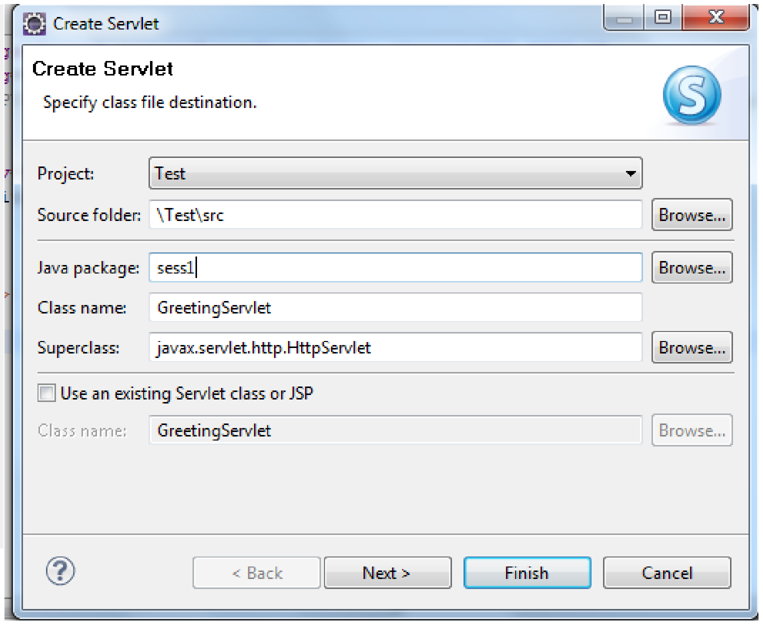
- Apache Tomcat v4.1
- Apache Tomcat v5.0
- Apache Tomcat v5.5
- Apache Tomcat v6.0
- Apache Tomcat v7.0
- Basic
- IBM
- JBoss
- ObjectWeb
- Oracle

Apache Tomcat v7.0 supports J2EE 1.2, 1.3, 1.4, and Java EE 5 and 6 Web modules.

- Open Eclipse JEE IDE and enter a workspace location.
- Select File→ New →Dynamic Web Project
- Enter the name of the project say “Test”.
- Select Target runtime as Apache Tomcat v.7.0. Select the Tomcat 7.0 installation directory.
- Leave the rest of the default entries as it is. Close the Welcome screen.

2. Create Servlet and override doGet()




Create Servlet

Create Servlet

Specify class file destination.

Project: Test

Source folder: \Test\src

Java package: sess1

Class name: GreetingServlet

Superclass: javax.servlet.http.HttpServlet

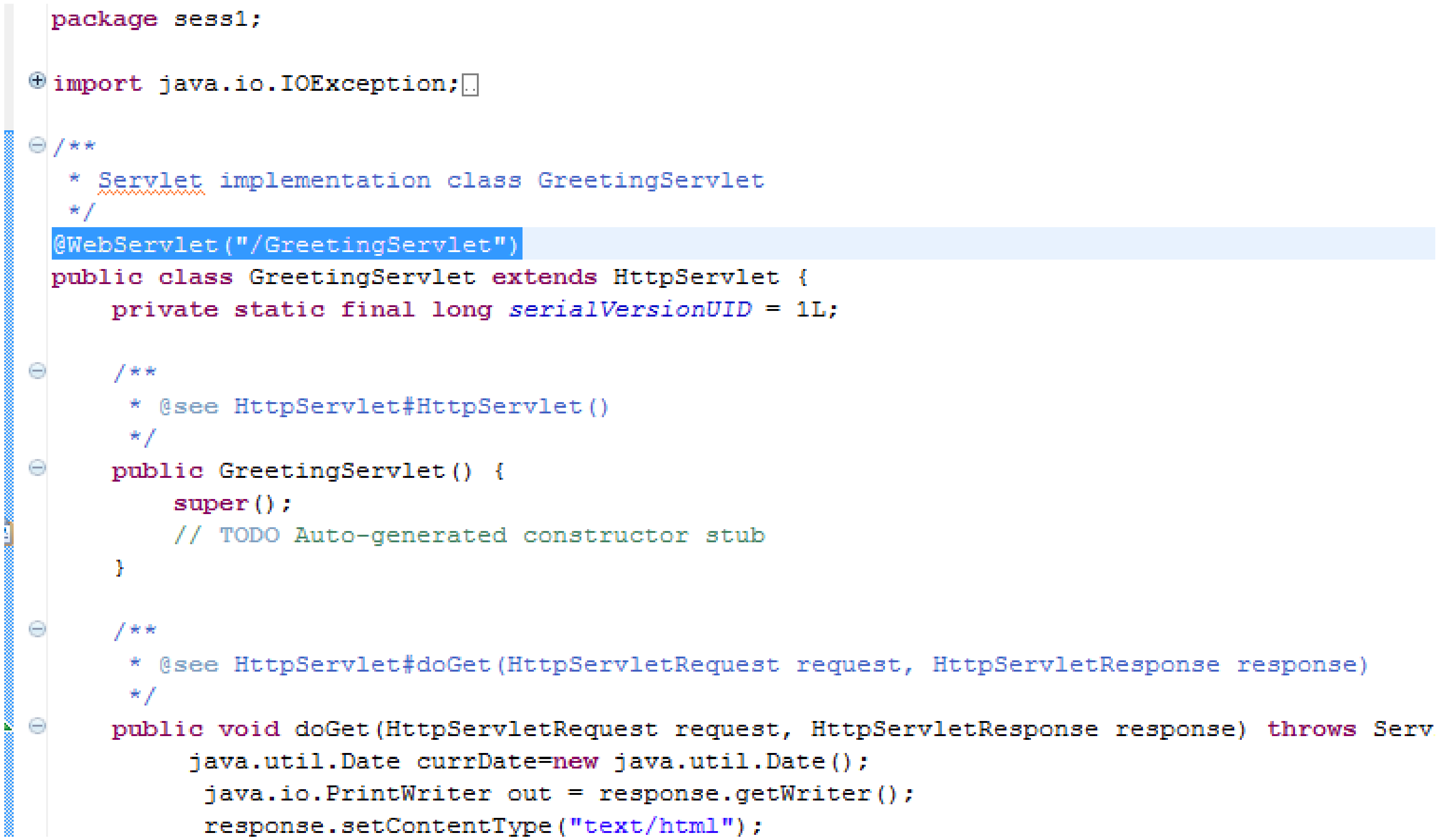
[ ] Use an existing Servlet class or JSP

Class name: GreetingServlet

- Right click on the project or Deployment Descriptor, and Select Servlet.
- Enter a package name say “sess1” and servlet name say “GreetingServlet”
- Enter the code in the `doGet()` method and make it `public`.

3. Map Servlet using annotation

- By default the code creates an annotation for mapping the servlet class to the URL that is going to used to access this component.
- The annotation used for this purpose is **`@WebServlet`**.



```
package sess1;

import java.io.IOException;

/**
 * Servlet implementation class GreetingServlet
 */
@WebServlet("/GreetingServlet")
public class GreetingServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public GreetingServlet() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws Serv
         java.util.Date currDate=new java.util.Date();
          java.io.PrintWriter out = response.getWriter();
          response.setContentType("text/html");
```

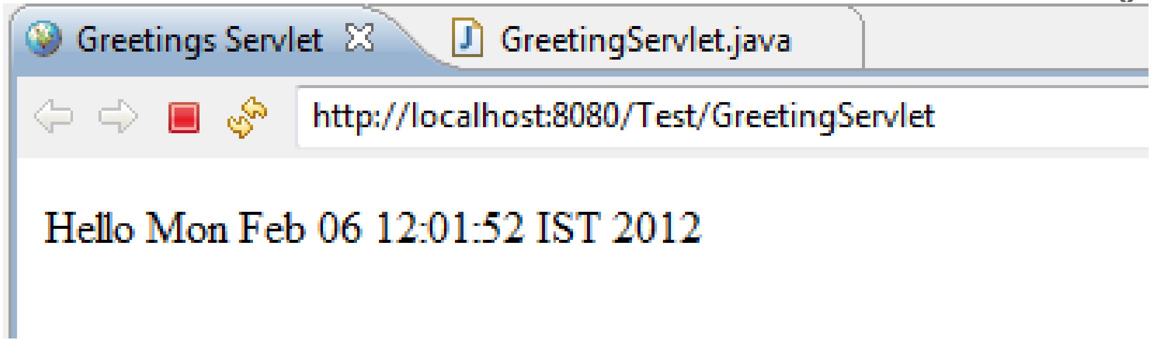
3. Map Servlet using web.xml

- Older web application used web.xml file for configuration.
- To use this, while creating project after entering the name for the project, click next and next again to get to the "web module" window. Click the checkbox "generate web.xml deployment descriptor" .
- web.xml is deployment descriptor is an xml file that helps in managing the configuration of an application. This is the file using which an application can communicate with the container and vice versa.

```
<web-app>
 <servlet>
    <servlet-name>Greet</servlet-name>
    <servlet-class>sess1.GreetingServlet</servlet-class>
 </servlet>
 <servlet-mapping>
    <servlet-name>Greet</servlet-name>
    <url-pattern>/GreetingServlet</url-pattern>
 </servlet-mapping>
</web-app>
```

4. Deploy and test the application

- Clicking on the run icon opens up a dialog for deploying the application on the selected server.
- Since we have only one server configured, we select that server.
- Click next and finish.
- Tomcat 7.0 server starts.
- The default browser window of the eclipse also opens.
- The URL of the servlet is http://servername:8080/application_name/servlet_url.
- Note that we are able to access this servlet by specifying URL in the address because we have overridden doGet(). I



http://localhost:8080/Test/GreetingServlet

```
Hello Mon Feb 06 12:01:52 IST 2012
```

Activity: understanding GET and POST

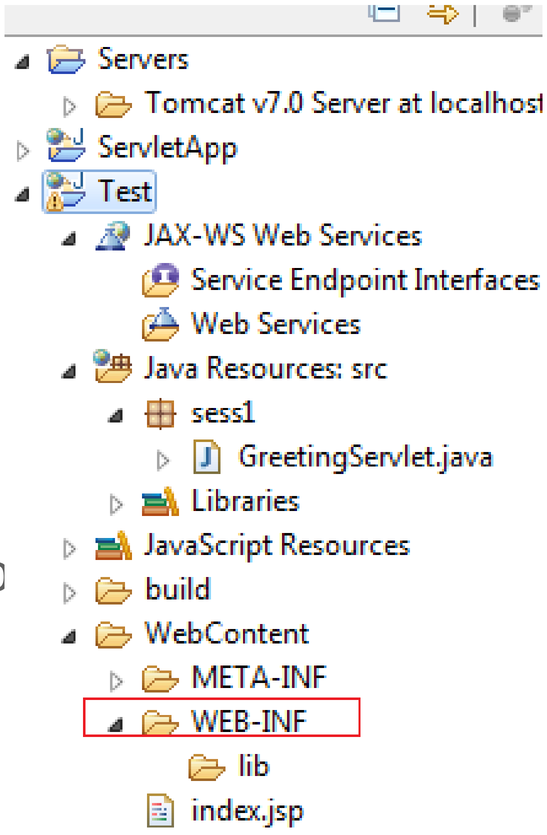
- Move the code in doGet() to doPost().
- Remove the doGet() method.
- Run the application and note the results.

Packaging of web application

- Packaging application means placing the files in the appropriate placeholders (directory).
- Servlet specification lays out the rules of how enterprise applications should be packaged.
- This is necessary so that the container (any j2ee container) knows where to find the files (the servlet class files, html class files etc.)
- Since a web application is composed of many files, the specification also tells us how to archive the files and deploy it as single file application.

Hierarchy

- The root of this hierarchy (document root, project name) is used as context path for web application by default.
- The application hierarchy has a folder named “WEB-INF”. This directory contains all things related to the application.
- This path is not directly accessible through browser. it is not part of the public document tree o the application.
- The web application is deployed as a WAR file (Web Application ARchive File) with extension as .war.



```
Servers
    Tomcat v7.0 Server at localhost
ServletApp
Test
    JAX-WS Web Services
        Service Endpoint Interfaces
        Web Services
    Java Resources: src
        sess1
            GreetingServlet.java
        Libraries
    JavaScript Resources
    build
    WebContent
        META-INF
        WEB-INF
            lib
        index.jsp
```

Contents of WEB-INF

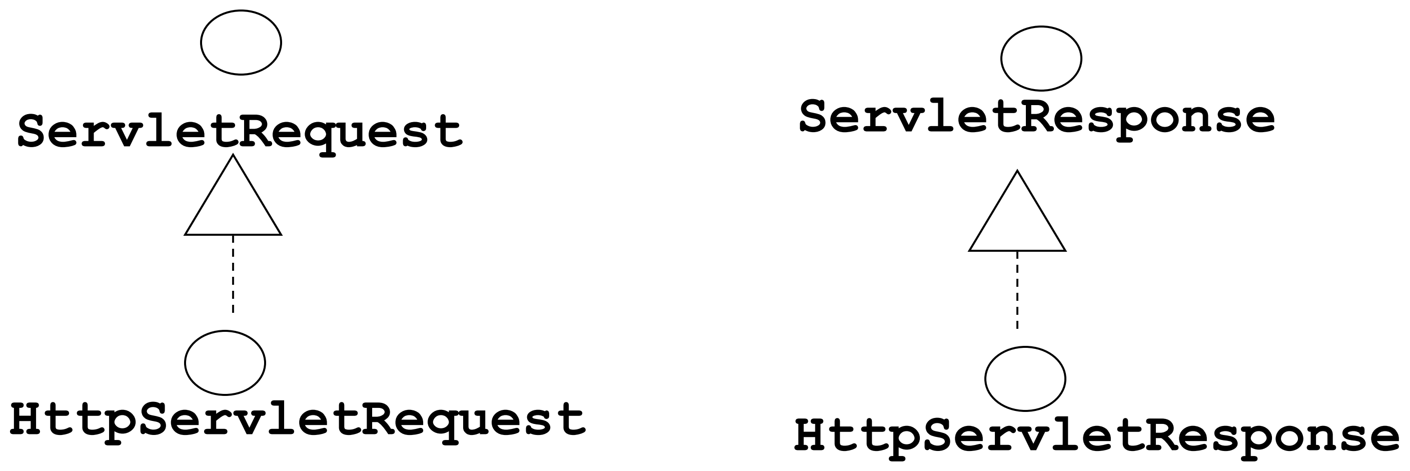
- The /WEB-INF/web.xml deployment descriptor.
- The /WEB-INF/classes/ directory for servlet and utility classes. The classes in this directory must be available to the application class loader.
- The /WEB-INF/lib/*.jar area for Java ARchive files. These files contain servlets, beans, and other utility classes useful to the Web application. The Web application class loader must be able to load classes from any of these archive files.

Example of a typical web application:

```
/index.html
/register.jsp
/images/logo.gif
/WEB-INF/web.xml
/WEB-INF/lib/classes12.jar
/WEB-INF/classes/sess1/RegisterServlet.class
```

Here the Servlet class is `sess1.RegisterServlet`

Request and Response hierarchy



- **ServletRequest** and **ServletResponse** are interfaces. These are passed to the **service** method of **GenericServlet**
- **HttpServletRequest** and **HttpServletResponse** interface objects are passed to all the **service**() and **doXXX**() methods defined by **HttpServlet**

Some methods of `ServletRequest`

- **`BufferedReader getReader()`**
- **`ServletInputStream  getInputStream()`**

  Both the methods can be used to called to read the body of the request but both should not be called simultaneously. **`llegalStateException`** will be thrown if used simultaneously

- **`int getContentLength():`** length of the request body, -1 if the length is not known. For HTTP request this is the same value as returned by the variable CONTENT_LENGTH
- **`Enumeration getParameterNames():`** Returns an **`Enumeration`** of String objects containing the names of the request parameters contained or an empty Enumeration.
- **`String getParameter(String name)`**
- **`String[] getParameterValues(String name)`**

  Both the methods are used to get the form parameter values of HTTP or HTTPS request

  We will see using an example how to work with these.

Some methods of `HttpServletRequest`

- **`String getMethod():`** Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT
- **`String getHeader(String name)`**
- **`Enumeration getHeaderNames()`**

  Returns all the values and names of the specified request header as an Enumeration of String objects. The header name is case insensitive.
- **`String getQueryString():`** Returns the query string that is contained in the request URL after the path
- **`String getContextPath()`** , **`String getServletPath()`** and **`String getPathInfo()`**

  http://localhost:8080/Test/GreetingServlet → **`getContextPath()`** returns **`/Test`** and **`getServletPath`** returns **`/GreetingServlet`**

  `getPathInfo()` gets the rest of the path information after the servlet path in case there are any.

Some methods of `ServletResponse`

- **`void setContentType(String type):`** Sets the content type (MIME type) of the response being sent to the client.
- **`void setContentLength(int len):`**Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header.
- **`ServletOutputStream getOutputStream()`**
- **`PrintWriter getWriter()`**

These method is used to write be called to write to the response body. Both of these methods cannot be used simultaneously. **`IllegalStateException`** will be thrown if used simultaneously.

Some methods of `HttpServletResponse`

- Status code based on HTTP response is available as static constants in this class. Some of the common status code s are

  `SC_ACCEPTED (202), SC_BAD_REQUEST(400),`
  `SC_NOT_FOUND(404), SC_INTERNAL_SERVER_ERROR (500),`
  `SC_METHOD_NOT_ALLOWED(405), SC_REQUEST_TIMEOUT (408),`
  `SC_NOT_IMPLEMENTED (501), SC_SERVICE_UNAVAILABLE(503)`

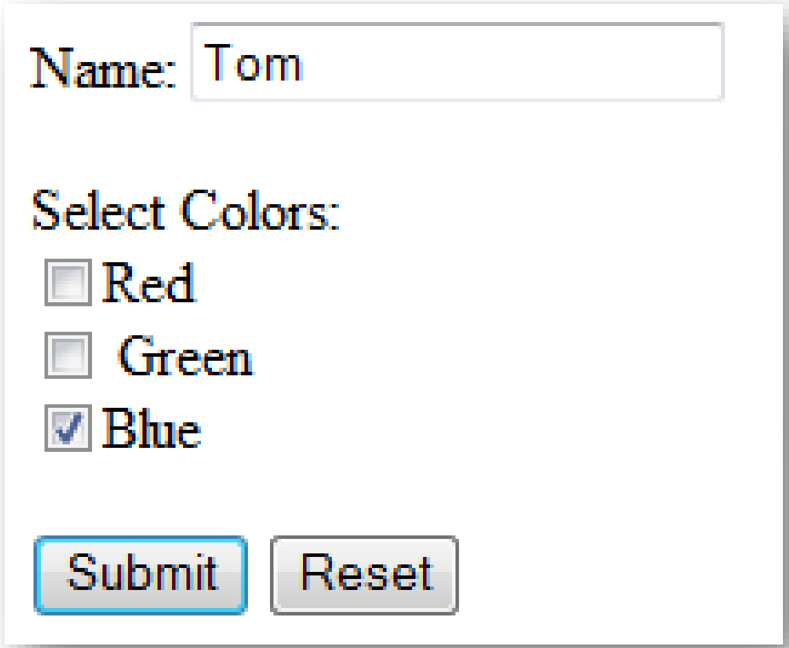
- **`String addHeader(String name,String value)`**
- **`void addIntHeader(java.lang.String name, int value)`**
- **`void addDateHeader(java.lang.String name, long date)`**

  The above methods add a response header with the given name and value

- **`void setStatus(int sc)`** : Sets the status code for response

Example : to get form data in the servlet

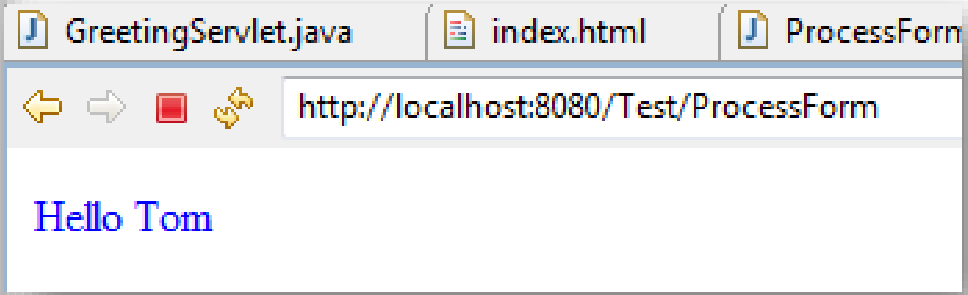
- In this example, we will create
  1. A HTML form that has a text field and checkbox field
  2. A servlet that fetches the data from the form and displays it.



Name: Tom

Select Colors:

- [ ] Red
- [ ] Green
- [x] Blue



http://localhost:8080/Test/ProcessForm

Hello Tom

index.html

- Create a HTML page by right clicking on the project. Name the page as index.html and enter the form inside the <body> as

```
<form method="POST" action="ProcessForm">
Name: <input type="text" name="uname">
<p>Select Colors:<br>
<input type="checkbox" name="color" value="Red">
Red <br>
<input type="checkbox" name="color" value="Green">
Green<br>
<input type="checkbox" name="color" value="Blue">
Blue</p>
<p><input type="submit" value="Submit" name="B1">
<input type="reset" value="Reset" name="B2"></p>
</form>
```

- Note how the checkboxes are grouped as one set by using the same name. Radio buttons can also be grouped similarly.

Servlet that gets the form data

- **`HttpServetRequest’s getParameter()`** method gets single value . This is used to get the uname textbox value of the HTML form.
- **`HttpServetRequest’s getParameterValuess()`** method gets multiple values in the form of array of String . Since multiple checkboxes can be selected, multiple values are passed. This is can be obtained using **`getParameterValuess()`** method.

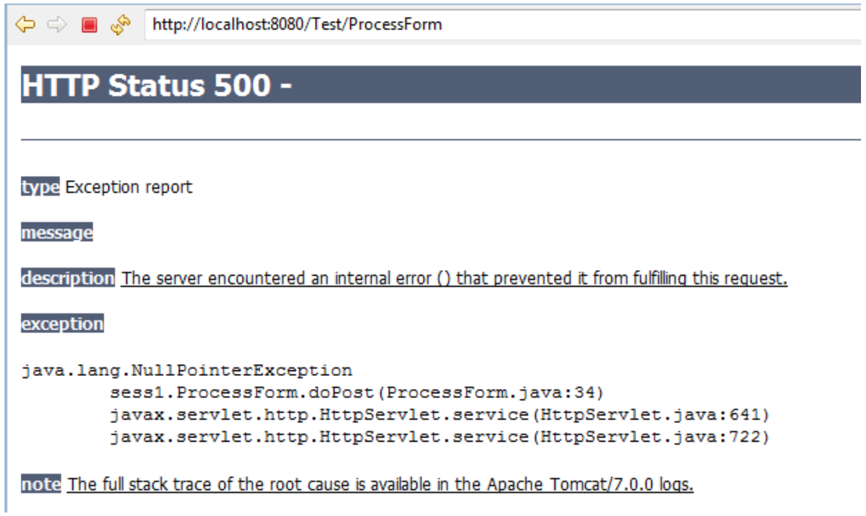
Servlet to get multiple param

```
// imports here
@WebServlet("/ProcessForm")
public class ProcessForm extends HttpServlet {
protected void doPost(HttpServletRequest request,
   HttpServletResponse response) throws ServletException,
   IOException {
java.io.PrintWriter out=response.getWriter();
out.println("<html>");
String name=request.getParameter("uname");
String colors[]=request.getParameterValues("color");

for(int i=0;i<colors.length;i++)
out.println("<font color='"+ colors[i]+"'>Hello
   "+name+"</font><br>"  );
out.println("</body></html>");
}
```

Tell me what

- What happens if you don't select any of the checkboxes?
- If a request parameter does not exists null is returned.
- If checkboxes are not selected, the response packet will not have request parameter "color".
- Therefore a **`NullPointerException`** is thrown.
- When a servlet throws a runtime exception, we end up getting an error page with HTTP Status 500.



http://localhost:8080/Test/ProcessForm

HTTP Status 500 -

type Exception report

message

description The server encountered an internal error () that prevented it from fulfilling this request.

exception

```
java.lang.NullPointerException
	sess1.ProcessForm.doPost(ProcessForm.java:34)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:641)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:722)
```

note The full stack trace of the root cause is available in the Apache Tomcat/7.0.0 logs.

Test your understanding

- What will happen if you type http://localhost:8080/Test/ProcessForm in the address bar?

`SingleThreadModel` Interface

- In the servlet model that we have seen so far, a single servlet processes multiple requests simultaneously.
- This means that the **`doGet`** and **`doPost`** methods must be careful to synchronize access to fields and other shared data, since multiple threads may be trying to access the data simultaneously.
- On the other hand, you can have your servlet implement the **`SingleThreadModel`** interface, as below.
- **`public class YourServlet extends HttpServlet implements SingleThreadModel {...}`**
- This usage is highly discouraged because it makes the application less scalable.

Test your understanding

```
//assume imports
public class Getdate extends HttpServlet{
java.util.Date currDate=new java.util.Date();
public void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
 java.util.Date now=new java.util.Date();
java.io.PrintWriter out = response.getWriter();
response.setContentType("text/html");

out.println("<html><head><title>Greetings Servlet
</title></head>");
out.println("<body>"+ currDate.compareTo(now)
+"</body></html>");
}
}}
```

Will the code always print 0?

Tell me what

- We must be careful about the variable declaration in the servlet because they are not thread-safe. But we do have a init() method that is used for servlet initialization, which means there do have cases where these members are declared. What kind of variables are declared in servlet?

  - Servlet should not have member variables whose value will change or depend upon user requests.
  - Servlet member variables can be used to assign things which are fixed like application configuration, resource configuration information etc.