Lecture 2
Servlet Introduction and Basics

Lecture Agenda

Applied

1. Web Application Structure and Directories.
2. Code and test simple servlets.
3. Use the web.xml file or annotation to map a servlet to a URL.
4. Provide for server-side data validation in your application.
5. Use the web.xml to set initialization parameters.
6. Use the web.xml file to implement custom error handling.
7. Write debugging data for a servlet (console or log file).

Lecture Agenda

Knowledge

1. The basic structure of servlets.
2. A simple servlet that generates plain text.
3. A servlet that generates HTML.
4. Using helper classes.
5. The servlet life cycle.
6. Servlet debugging techniques.

Lecture Agenda

Knowledge Continued …

7 Describe servlets and servlet mapping.

8 Describe how parameters are passed to a servlet with HTTP GET.

9 List 3 reasons for using the HTTP POST method instead of HTTP GET.

10 Describe how **ServletContext** is used to get the path for a file.

11 Describe **init**(), **doGet**(), **doPost**(), and **destroy**() servlet methods.

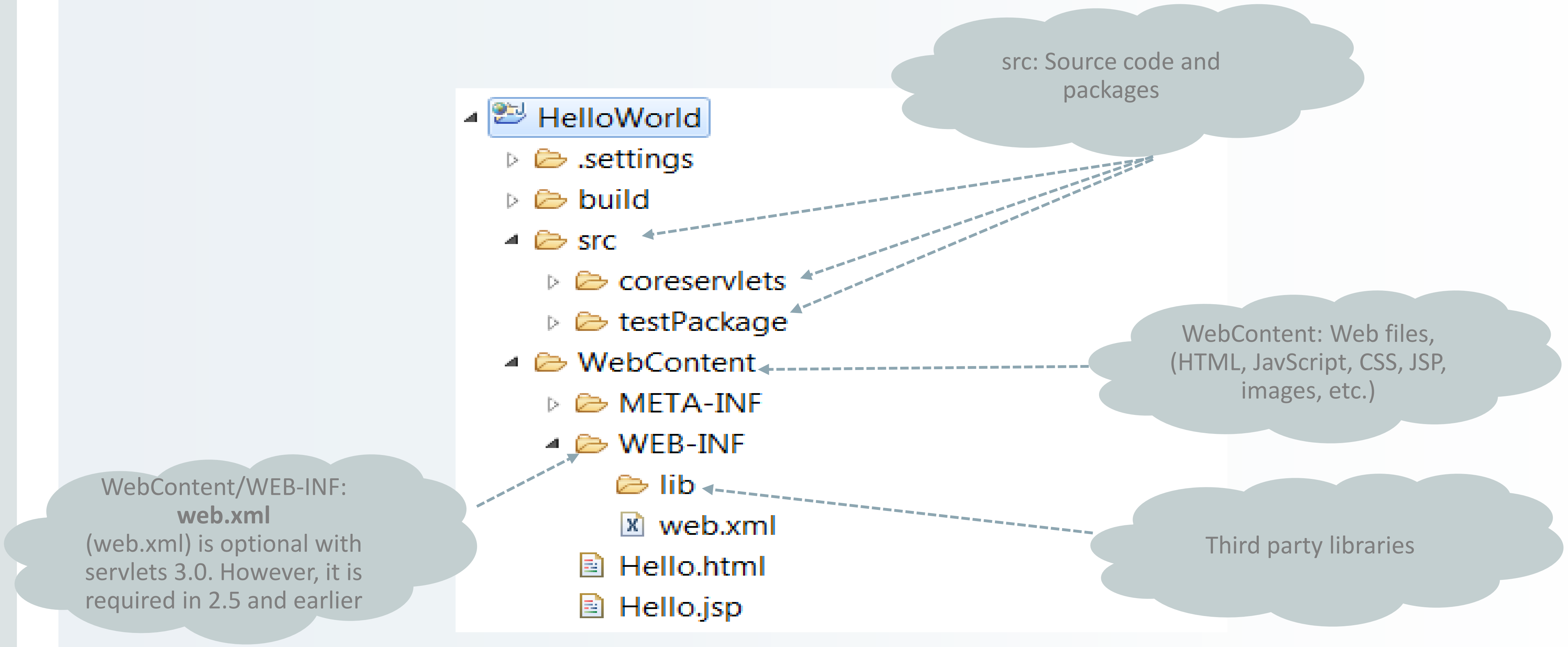
12 Explain why you should never use instance variables in servlets.

13 Describe the use of debugging data written to the console or log file.

Web Application Directory Structure

Web Application Directory Structure

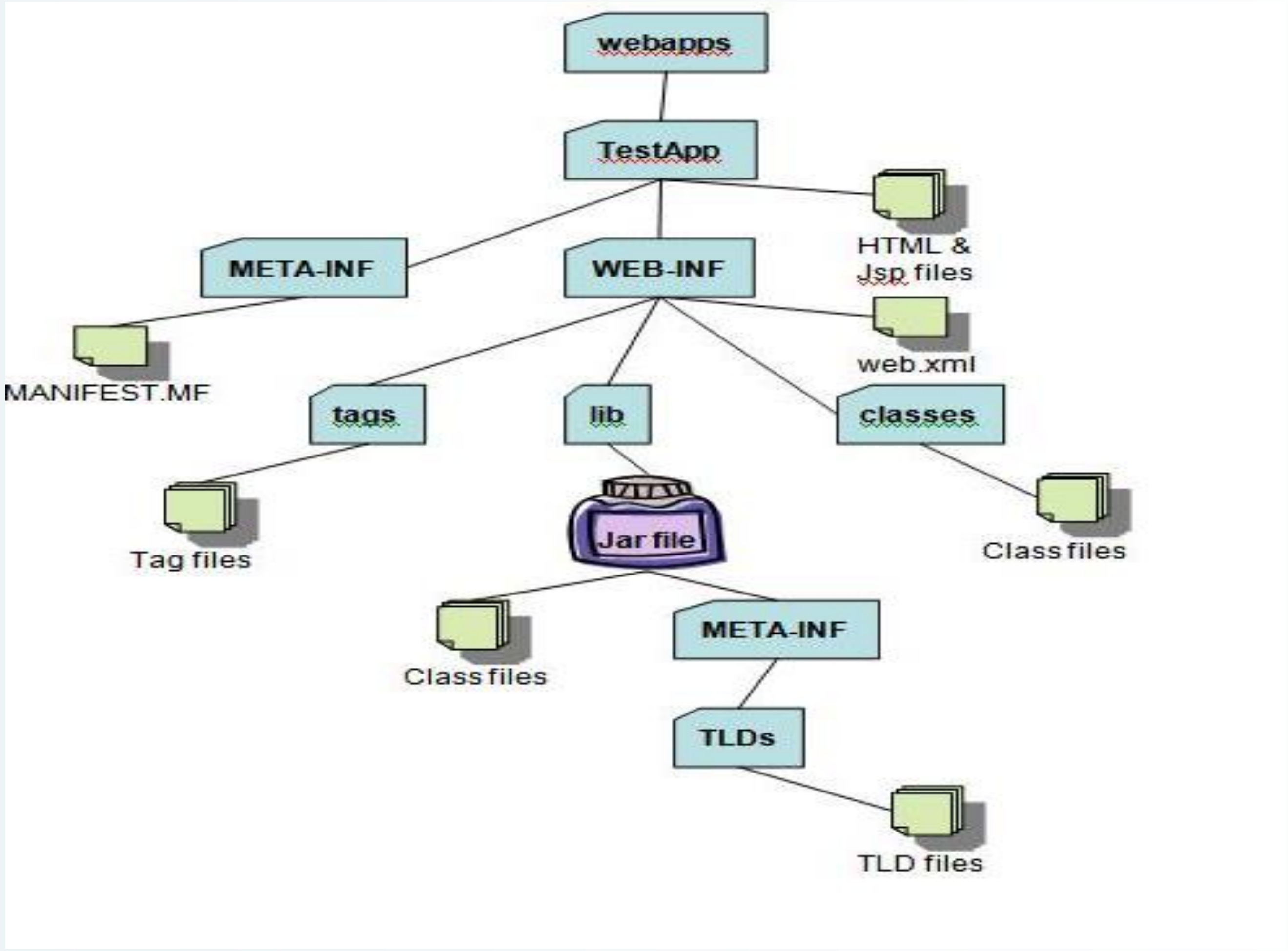
Eclipse directory structure



HelloWorld
.settings
build
src
coreservlets
testPackage
WebContent
META-INF
WEB-INF
lib
web.xml
Hello.html
Hello.jsp
src: Source code and packages
WebContent: Web files, (HTML, JavScript, CSS, JSP, images, etc.)
Third party libraries
WebContent/WEB-INF: **web.xml**
(web.xml) is optional with servlets 3.0. However, it is required in 2.5 and earlier

Web Application Directory Structure

Tomcat directory structure

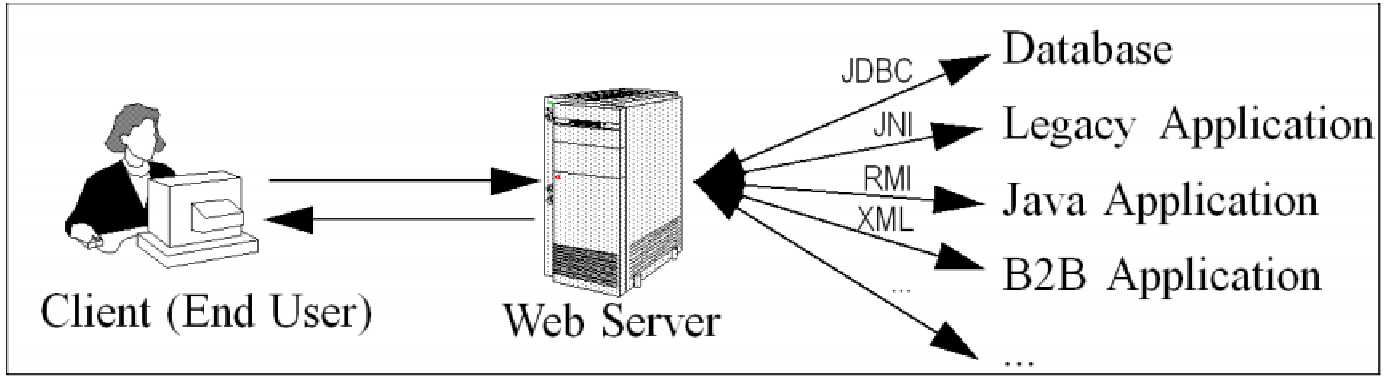


The Directories and Files for a Web Application

Directory	Description
root	Contains the HTML and JSP files
\WEB-INF	Contains the web.xml file and is not directly accessible from the web.
\WEB-INF\classes	Contains the servlets and other Java classes for your application. Each subdirectory corresponds with the package for the Java class.
\WEB-INF\lib	Contains any JAR files that contain Java class libraries that are used by the web application.
\META-INF	Contains the context.xml file that configures the web application.

Servlet Basics

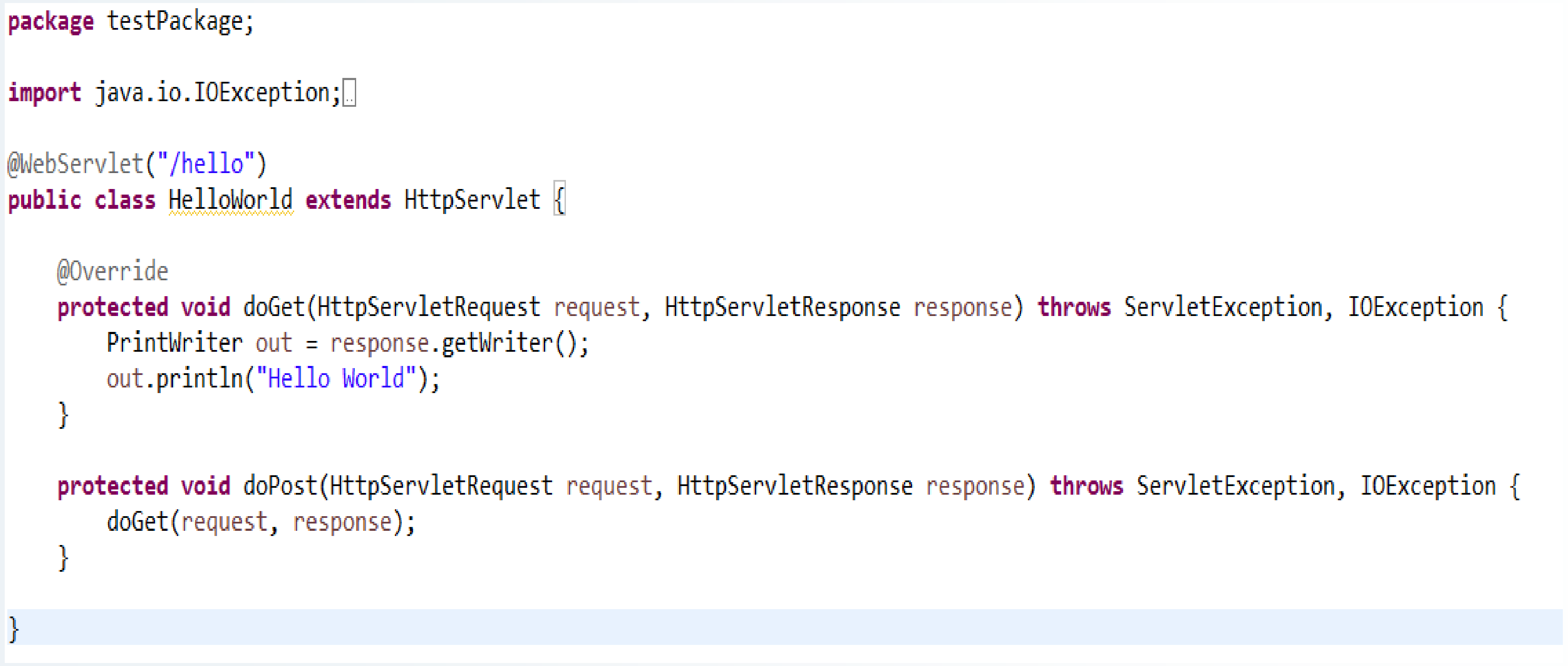
A Servlets Job



- Read explicit data sent by the client (form data).
- Read implicit data sent by the client (request headers).
- Generate the results.
- Send the explicit data back to the client (HTML).
- Send the implicit data back to the client (status codes / response headers).

Servlet that Generates Plain Text

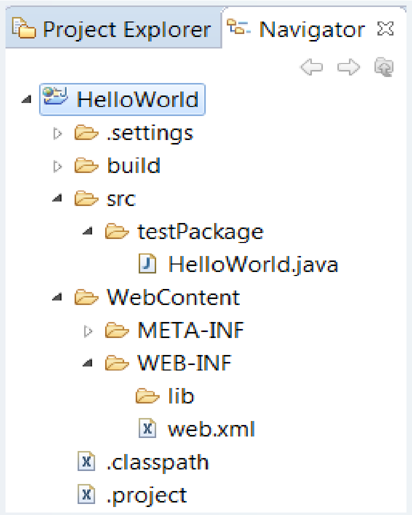
HelloWorld.java

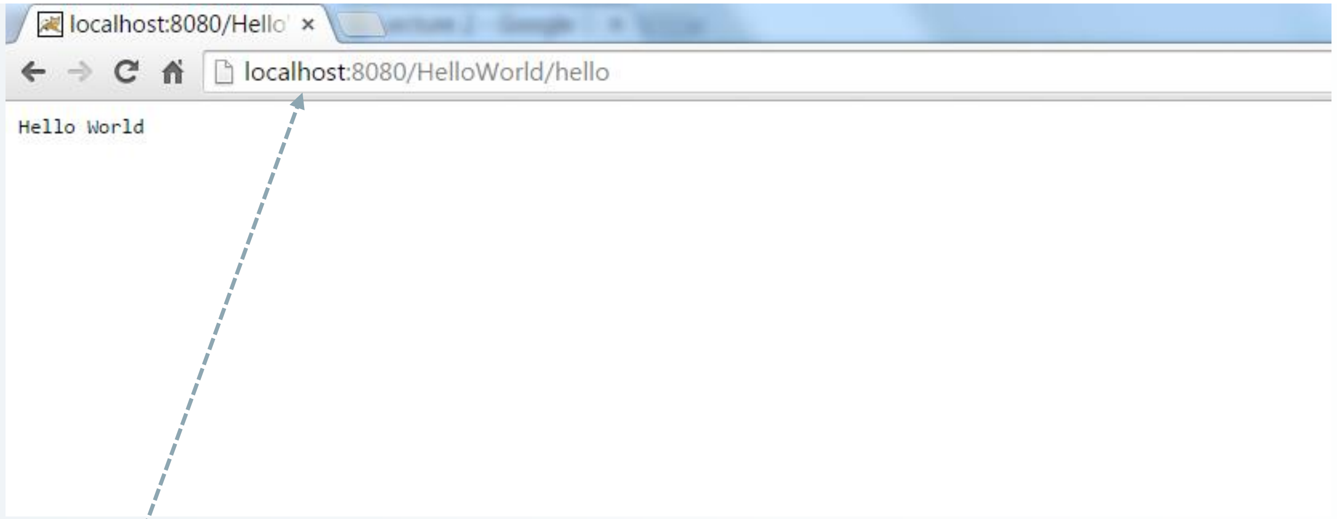


```
package testPackage;

import java.io.IOException;

@WebServlet("/hello")
public class HelloWorld extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("Hello World");
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }

}
```

Servlet that Generates Plain Text

HelloWorld.java





```
Hello World
```

- URL assumes project name is “HelloWorld”
- URL form http://*hostname:port/appname/servletname*

Interpreting HelloWorld Servlet

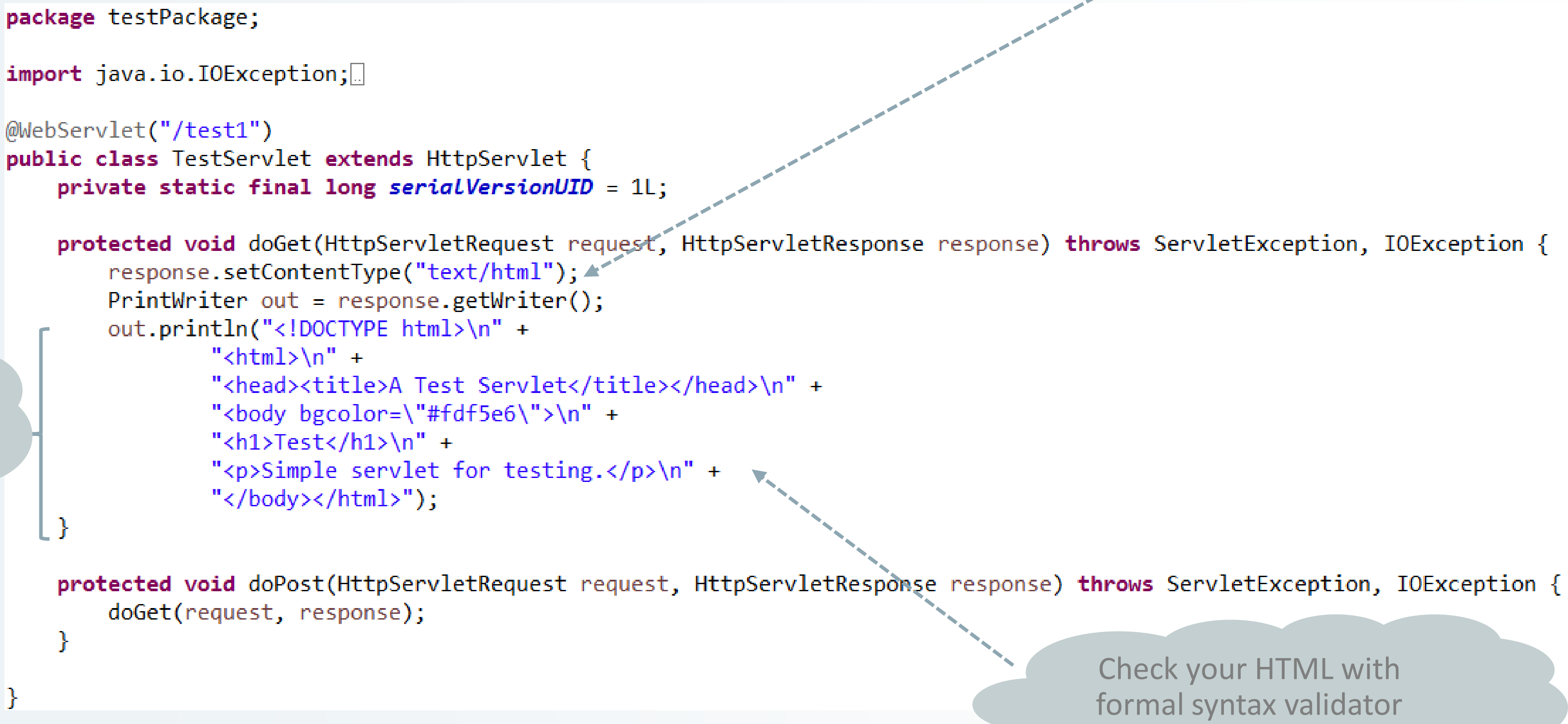
Code Concepts

- @WebServlet("/address")
  - This is the URL relative to the application name.
- doGet()
  - Code for an HTTP GET request.
- doPost()
  - Code for an HTTP POST request.
- HttpServletRequest
  - Contains any data that comes from the browser.
- HttpServletResponse
  - Used to send information back to the browser.
- @Override
  - General Best Practice when overriding inherited methods.

Servlet that Generates HTML

TestServlet.java

Tell the browser your returning HTML



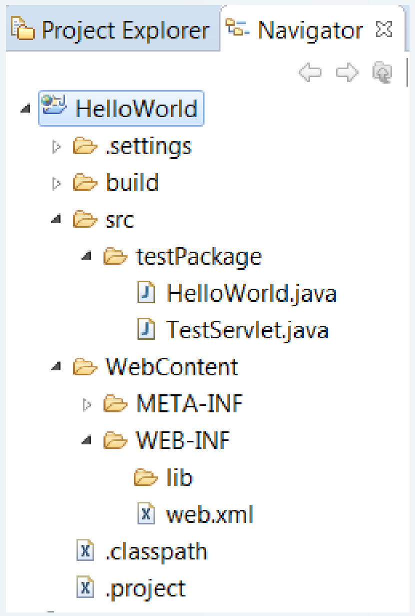
```java
package testPackage;

import java.io.IOException;

@WebServlet("/test1")
public class TestServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html>\n" +
                "<html>\n" +
                "<head><title>A Test Servlet</title></head>\n" +
                "<body bgcolor=\"#fdf5e6\">\n" +
                "<h1>Test</h1>\n" +
                "<p>Simple servlet for testing.</p>\n" +
                "</body></html>");
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }

}
```

Modify println statements to build a legal Web Page

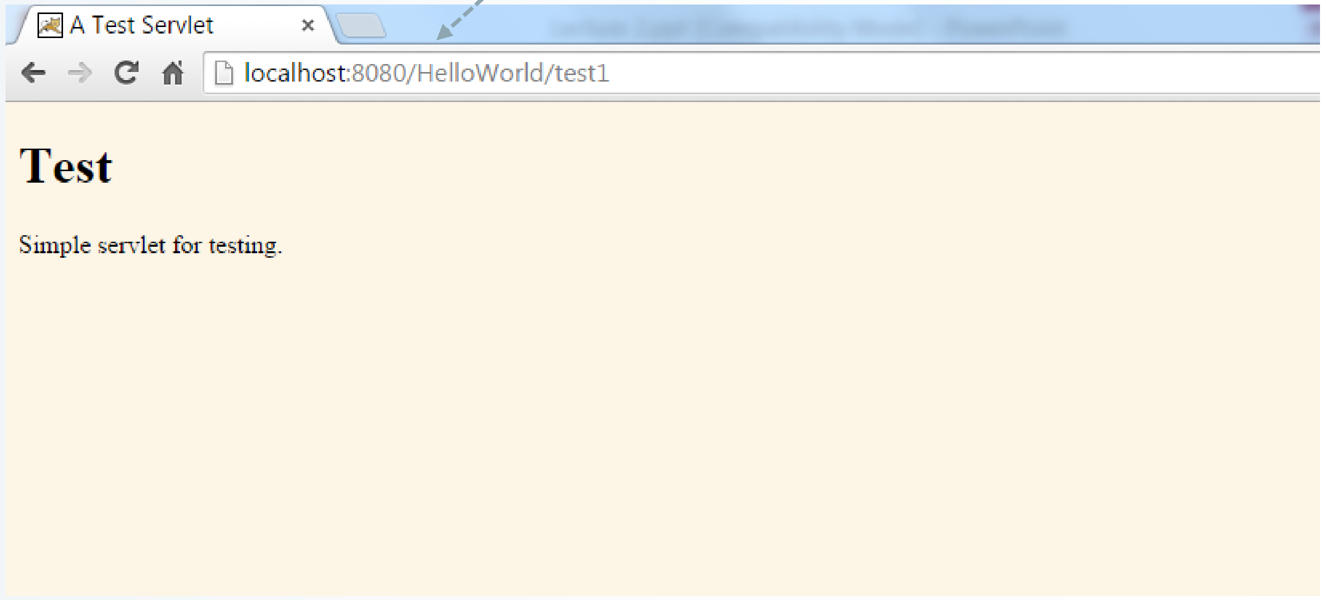
Check your HTML with formal syntax validator (http://validator.w3.org)

Servlet that Generates HTML

TestServlet.java



Project and Directory Structure



Test

Simple servlet for testing.

URL assumes project name is “HelloWorld” and servlet is named “test1”

Using Helper Classes

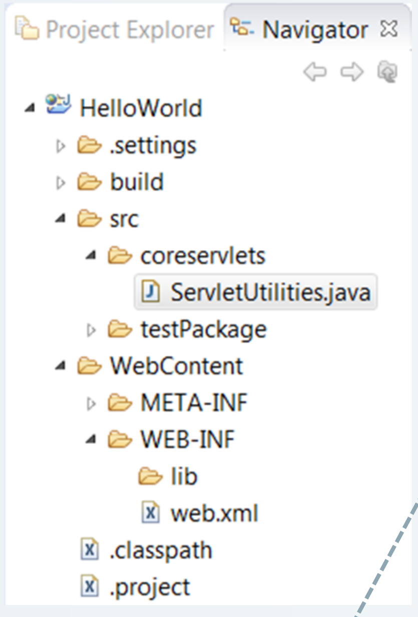
Using Helper Classes

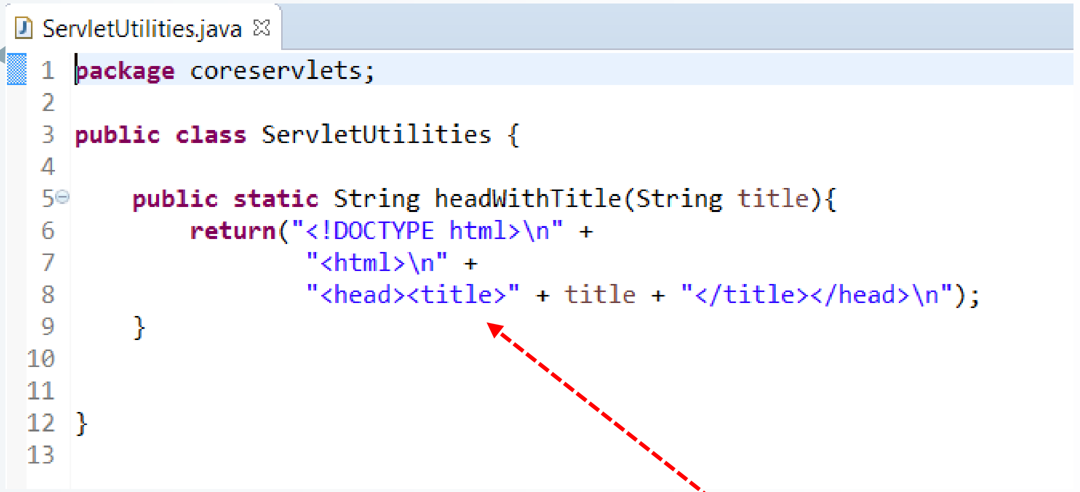
Theory

- All java code goes in the same place.
  - In Eclipse, source code goes in src/<*package name*>
- Always remember OO principles.
  1. Encapsulation
  2. Data abstraction
  3. Polymorphism
  4. Inheritance

HTML-Building Utility

Practical Example





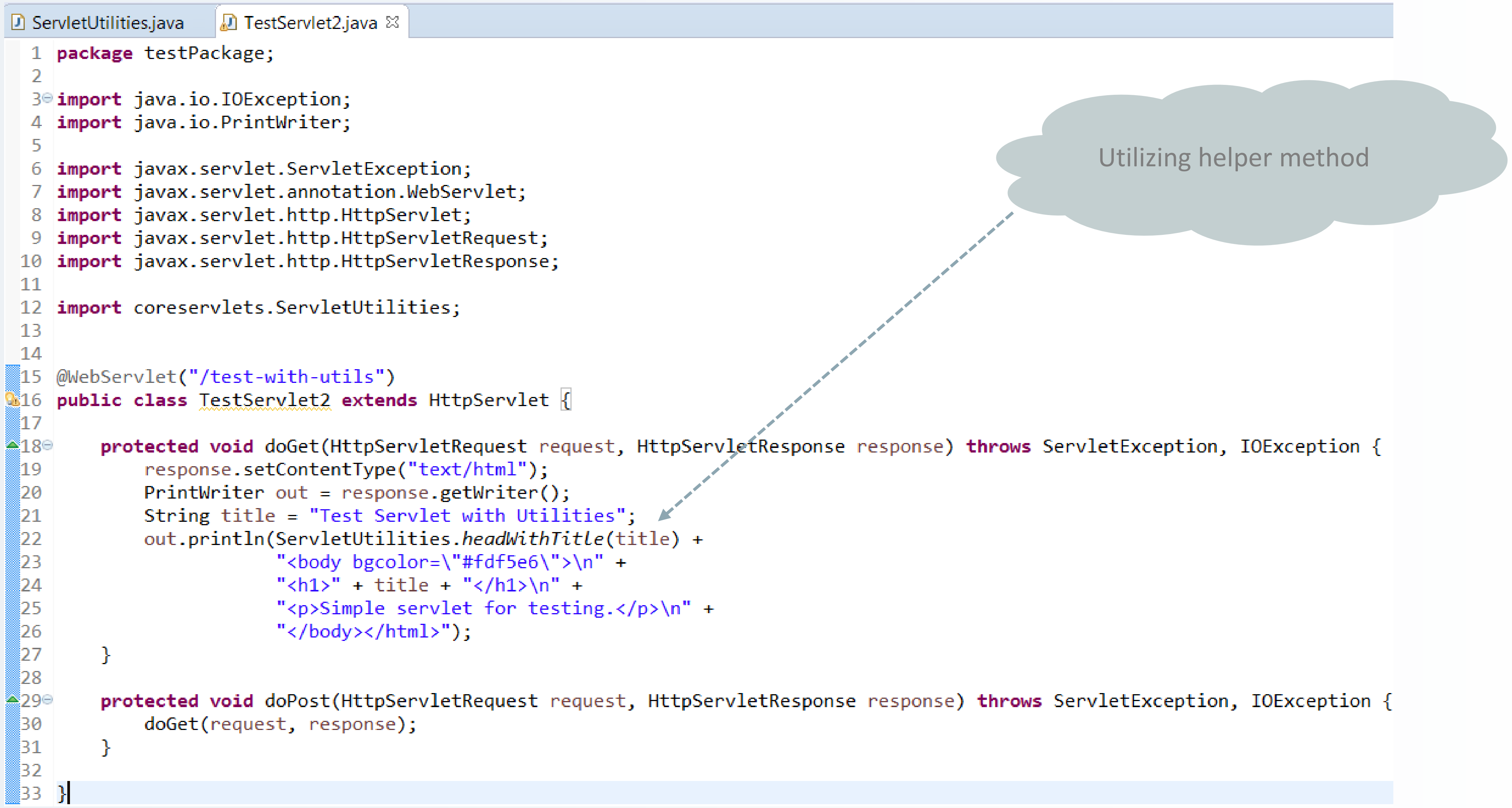
```java
package coreservlets;

public class ServletUtilities {

    public static String headWithTitle(String title){
        return("<!DOCTYPE html>\n" +
                "<html>\n" +
                "<head><title>" + title + "</title></head>\n");
    }


}

```

ServletUtilities class in this example, helps avoid repeating logic.

Don't go overboard, complete HTML packages (helpers) forwarding HTML content work poorly. Using JSPs is a better approach

Servlet Utilizing Helper

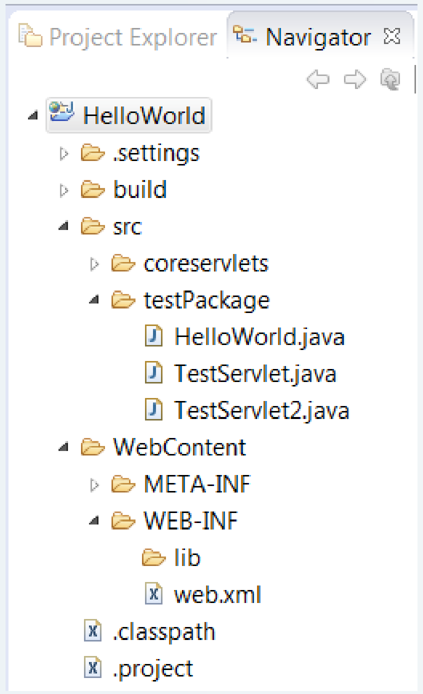
TestServlet2



```java
package testPackage;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import coreservlets.ServletUtilities;


@WebServlet("/test-with-utils")
public class TestServlet2 extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Test Servlet with Utilities";
        out.println(ServletUtilities.headWithTitle(title) +
                    "<body bgcolor=\"#fdf5e6\">\n" +
                    "<h1>" + title + "</h1>\n" +
                    "<p>Simple servlet for testing.</p>\n" +
                    "</body></html>");
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }

}
```

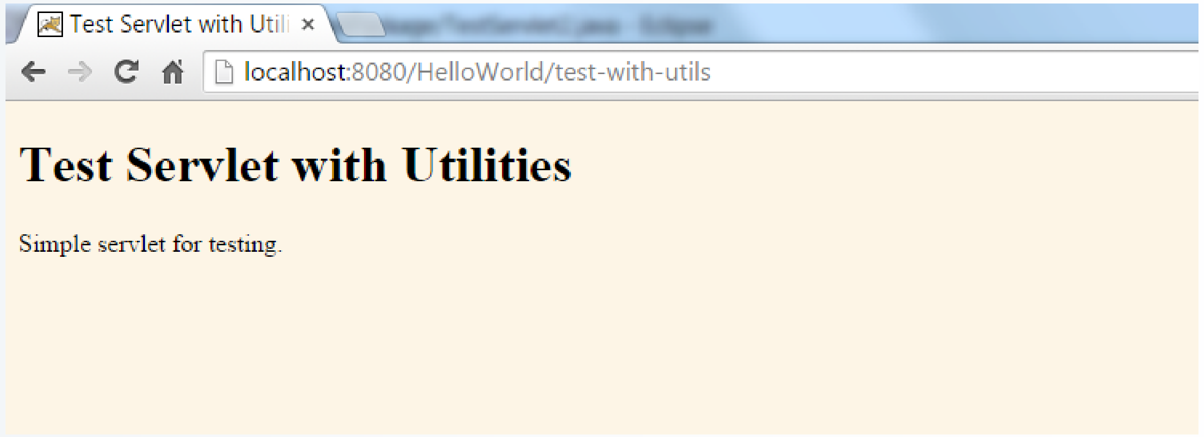
Servlet that Generates HTML

TestServlet.java


Project Explorer Navigator

```
HelloWorld
  .settings
  build
  src
    coreservlets
    testPackage
      HelloWorld.java
      TestServlet.java
      TestServlet2.java
  WebContent
    META-INF
    WEB-INF
      lib
      web.xml
  .classpath
  .project
```

Project and Directory Structure


Test Servlet with Utili

localhost:8080/HelloWorld/test-with-utils

Test Servlet with Utilities

Simple servlet for testing.

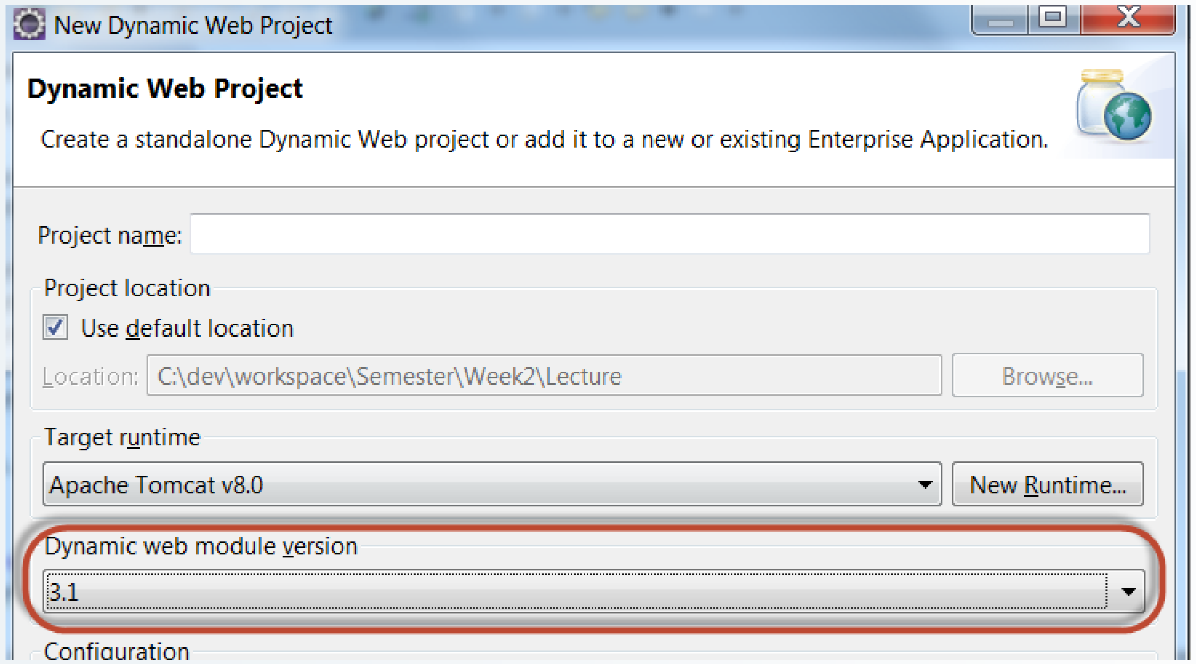
URL assumes project name is “HelloWorld” and servlet is named “test-with-utils”

Custom URLs and Annotations

Annotations

Annotation are supported in Tomcat 7 and Tomcat 8

Tomcat 7 has support for Servlet 3.0


New Dynamic Web Project

Dynamic Web Project

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

Project name:

Project location

☑ Use default location

Location: C:\dev\workspace\Semester\Week2\Lecture

Browse...

Target runtime

Apache Tomcat v8.0

New Runtime...

Dynamic web module version

3.1

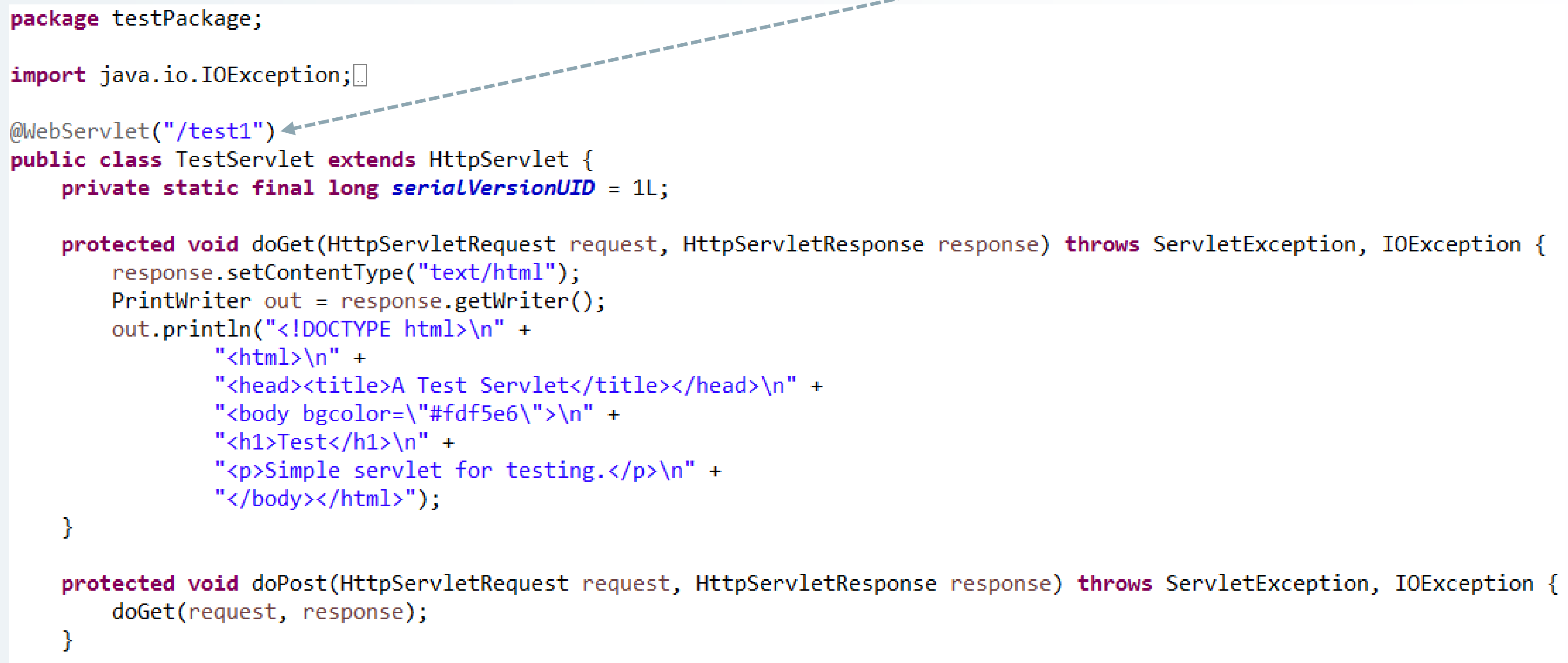
Configuration

Tomcat 8 has support for servlet 3.1

Annotations

@WebServlet annoation

This servlet is utilizing the **@WebServlet** annotation



```java
package testPackage;

import java.io.IOException;

@WebServlet("/test1")
public class TestServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html>\n" +
                "<html>\n" +
                "<head><title>A Test Servlet</title></head>\n" +
                "<body bgcolor=\"#fdf5e6\">\n" +
                "<h1>Test</h1>\n" +
                "<p>Simple servlet for testing.</p>\n" +
                "</body></html>");
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }
```

Since we are using **Tomcat 8** we know we can use annotations for servlet mapping.

Custom URLs and Annotations

Background

- Give address with @WebServlet

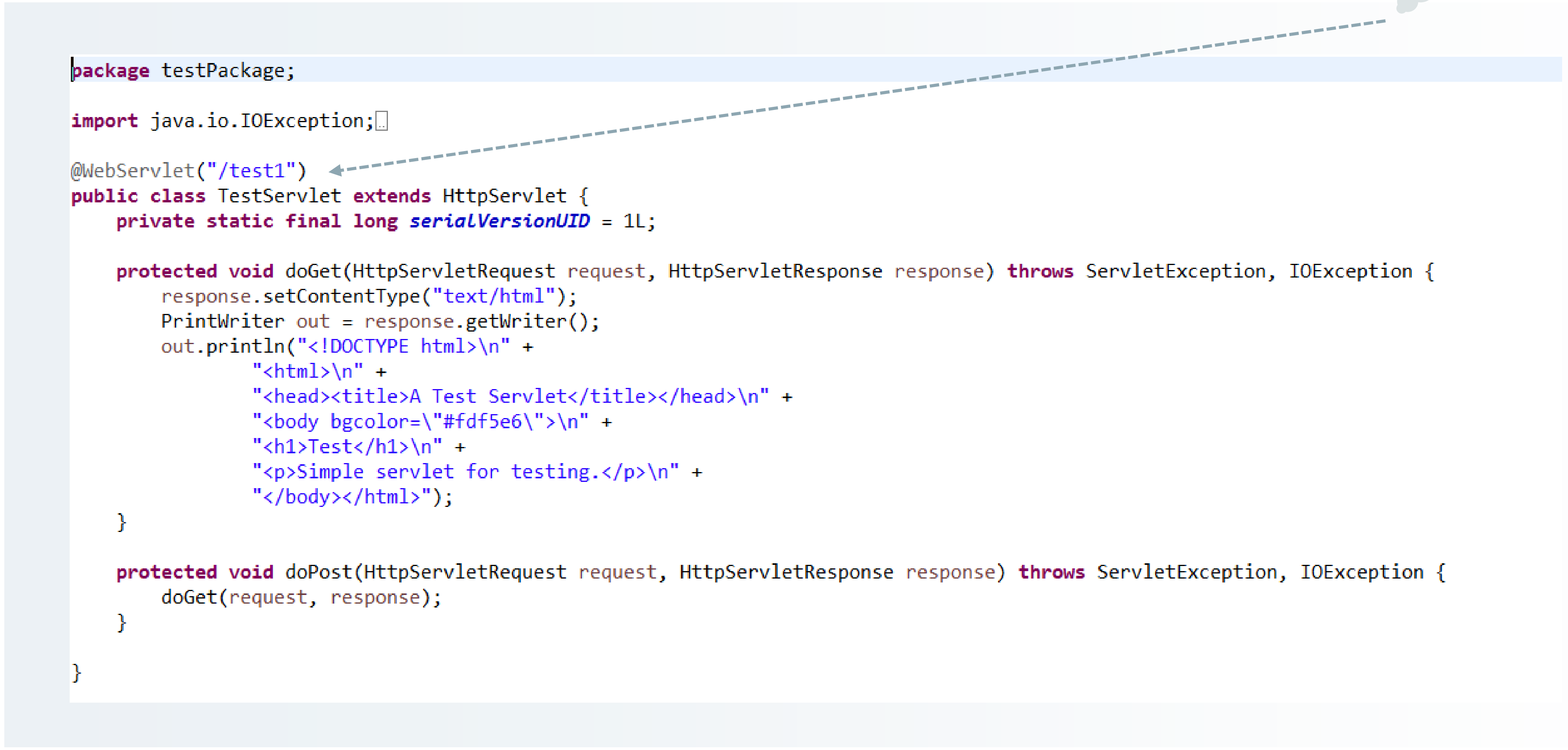
```
@WebServlet("/my-address")
Public class MyServlet extends HttpServlet { … }
```

- Resulting URL
  - http://*hostname*:*port*/*appname*/*my-address*
- Omit web.xml Entirely
  - web.xml can be used when using @WebServlet, but the entire file is completely optional.
  - In earlier servlet engine versions, web.xml was <u>mandatory</u>.

@WebServlet Annotations

More Examples

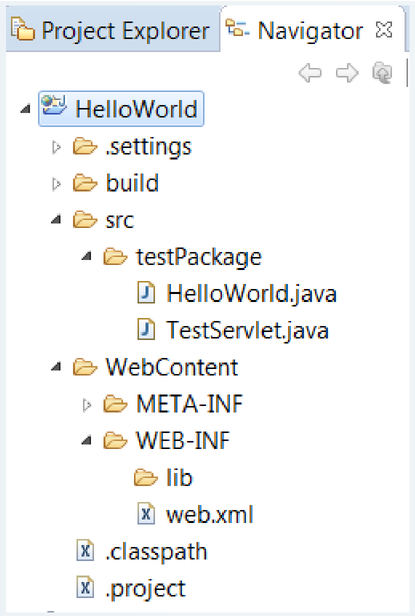
This servlet is utilizing the **@WebServlet** annotation. URL patter mapping “/test1”



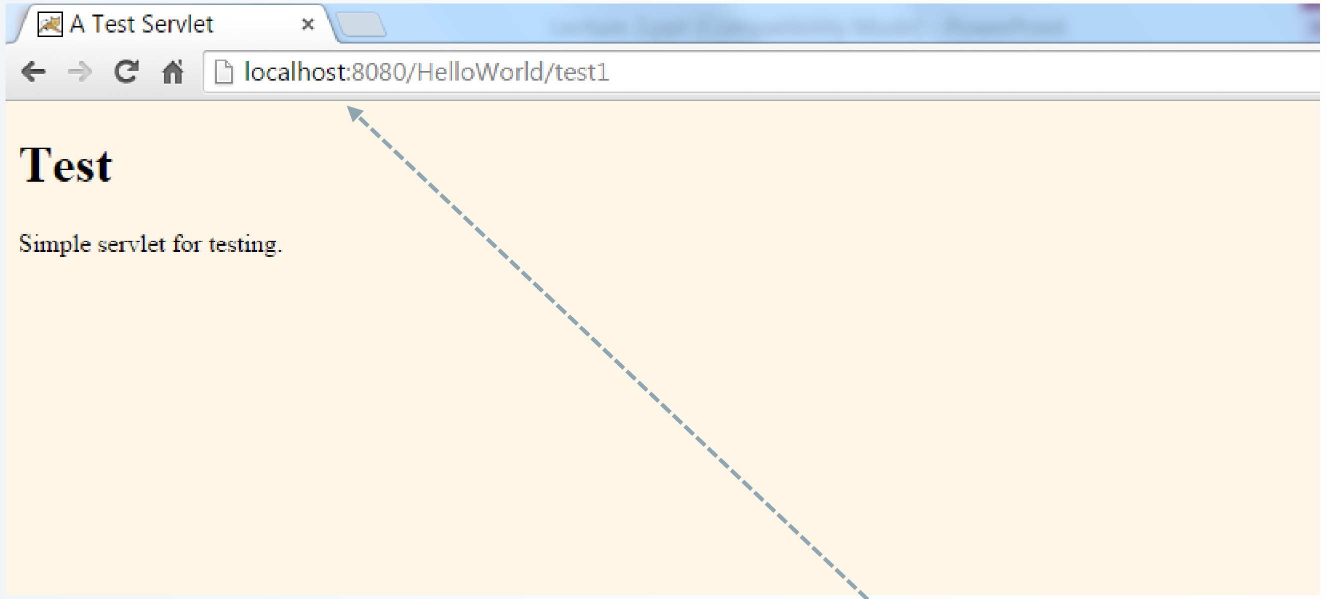
```
package testPackage;

import java.io.IOException;

@WebServlet("/test1")
public class TestServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html>\n" +
                "<html>\n" +
                "<head><title>A Test Servlet</title></head>\n" +
                "<body bgcolor=\"#fdf5e6\">\n" +
                "<h1>Test</h1>\n" +
                "<p>Simple servlet for testing.</p>\n" +
                "</body></html>");
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }

}
```

@WebServlet Annotations

More Examples



```
HelloWorld
  .settings
  build
  src
    testPackage
      HelloWorld.java
      TestServlet.java
  WebContent
    META-INF
    WEB-INF
      lib
      web.xml
  .classpath
  .project
```



Test

Simple servlet for testing.

Directory Structure

URL assumes project name is “HelloWorld”. Servlet name is “test1”.

What if were not using a Servlet 3.0+ compliant Servlet Engine?

Custom URLs and web.xml

Custom URLs and web.xml

Background

- Java Code

```
package myPackage
public class MyServlet extends HttpServlet { … }
```

- web.xml entry
  - Starting tags: <web-app> ….. </web-app>
- Name of Servlet

```
<servlet>
        <servlet-name>MyName</servlet-name>
        <servlet-class>myPackage.MyServlet<servlet-class>
</servlet>
```

Custom URLs and web.xml

Background continued …

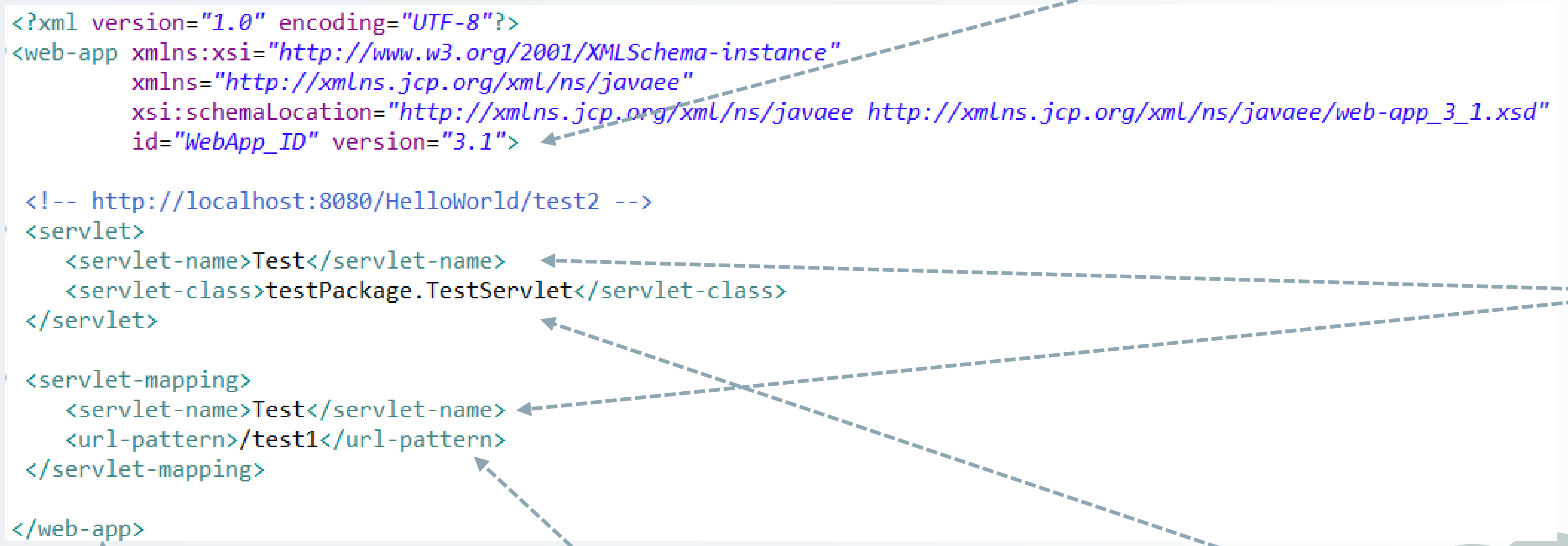
- Address (URL mapping) to servlet

```
<servlet-mapping>
        <servlet-name>MyName</servlet-name>
        <url-pattern>/my-address</url-pattern>
</servlet-mapping>
```

- Resultant Address
  – http://hostname:port/appname/my-address

web.xml

Defining custom URLs



```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         id="WebApp_ID" version="3.1">

  <!-- http://localhost:8080/HelloWorld/test2 -->
  <servlet>
     <servlet-name>Test</servlet-name>
     <servlet-class>testPackage.TestServlet</servlet-class>
  </servlet>

  <servlet-mapping>
     <servlet-name>Test</servlet-name>
     <url-pattern>/test1</url-pattern>
  </servlet-mapping>

</web-app>
```

Don’t edit this manually. The version should match the server version.

Arbitrary name but they **<u>must</u>** matched

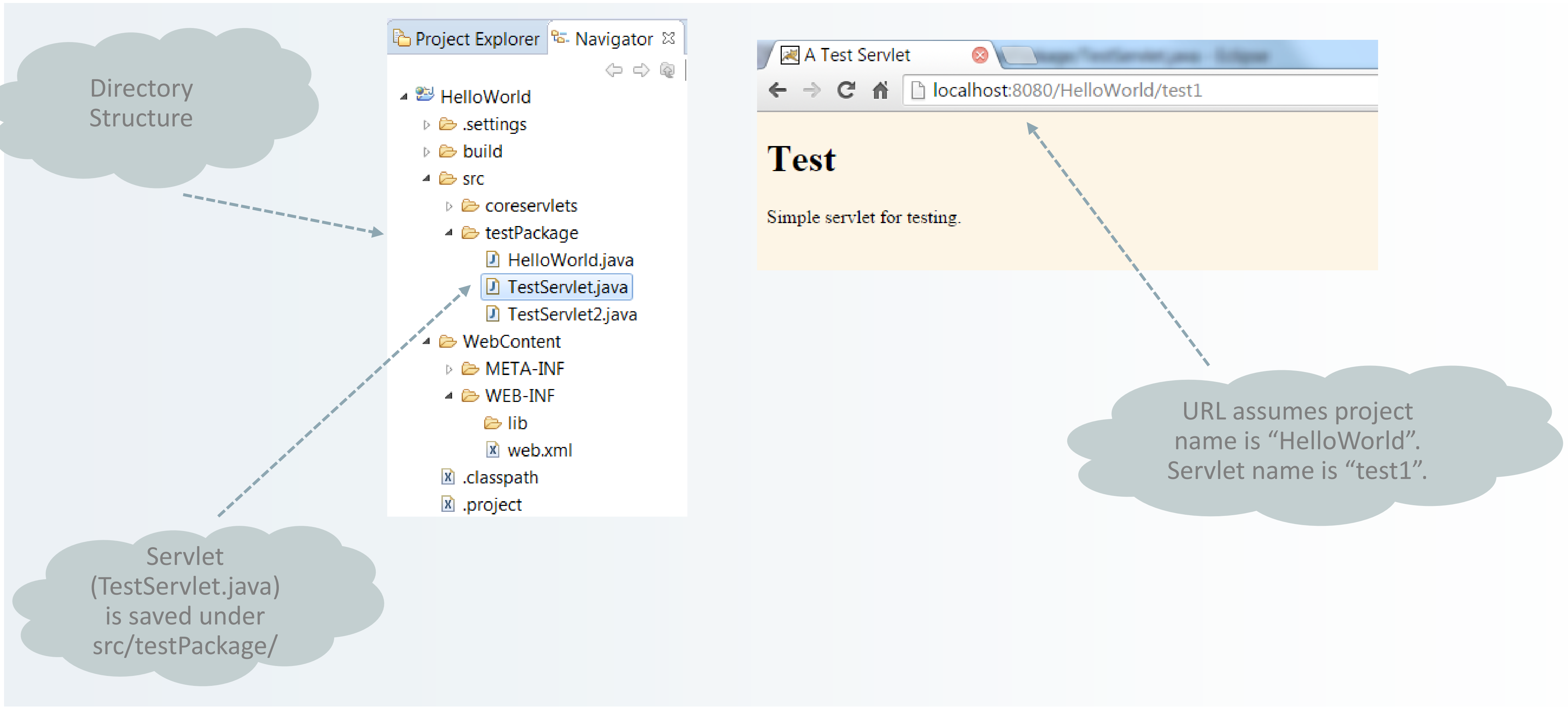
Fully qualified domain name

Note opening and closing tags (<web-app>)

The part of the URL that comes after the app (project) name. Should start with a forward slash.

Defining Custom URLs and web.xml

Result



Directory Structure

```
HelloWorld
  .settings
  build
  src
    coreservlets
    testPackage
      HelloWorld.java
      TestServlet.java
      TestServlet2.java
  WebContent
    META-INF
    WEB-INF
      lib
      web.xml
  .classpath
  .project
```

Servlet (TestServlet.java) is saved under src/testPackage/

localhost:8080/HelloWorld/test1

Test

Simple servlet for testing.

URL assumes project name is “HelloWorld”. Servlet name is “test1”.

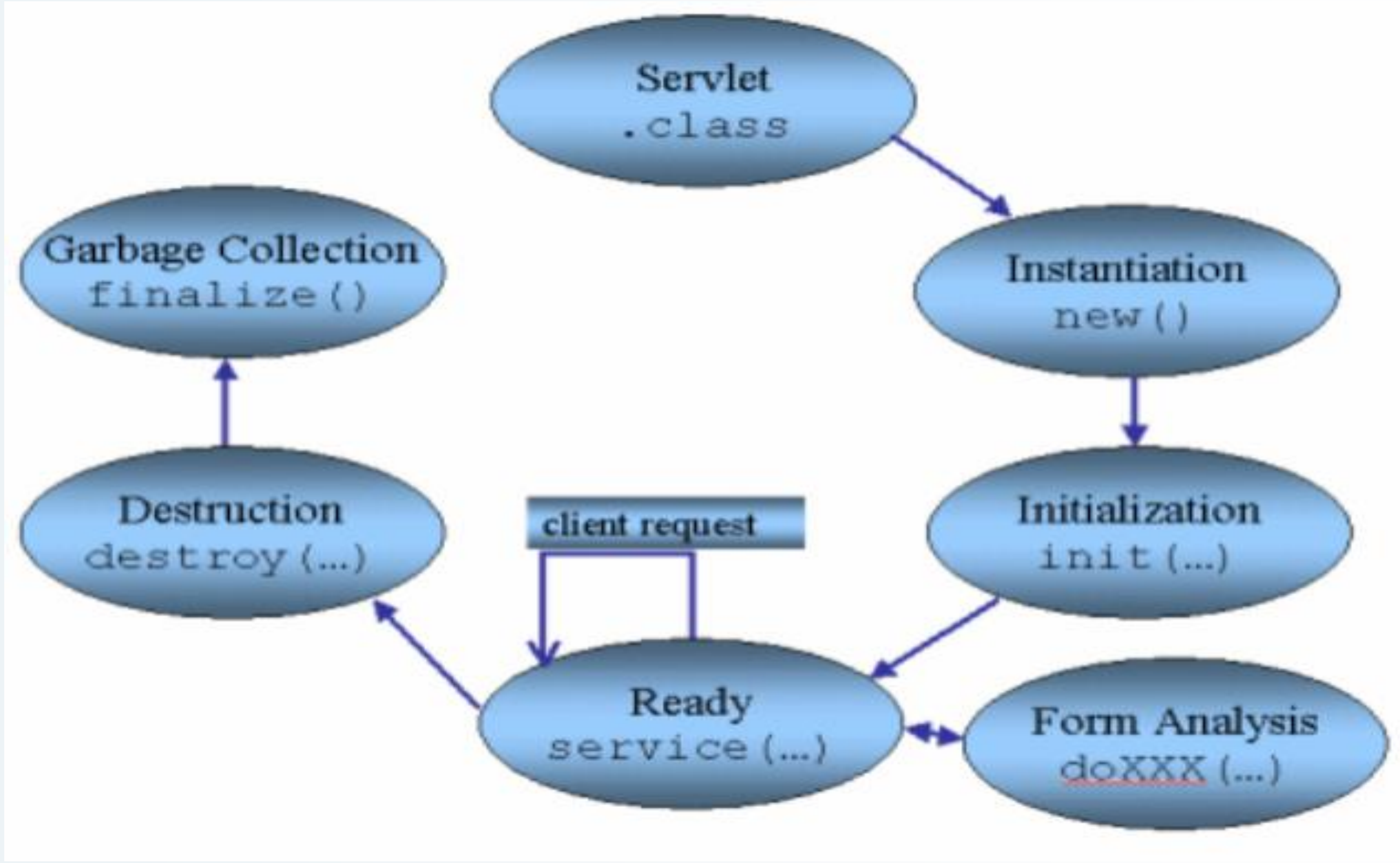
Advanced Topics

The Servlet Life Cycle

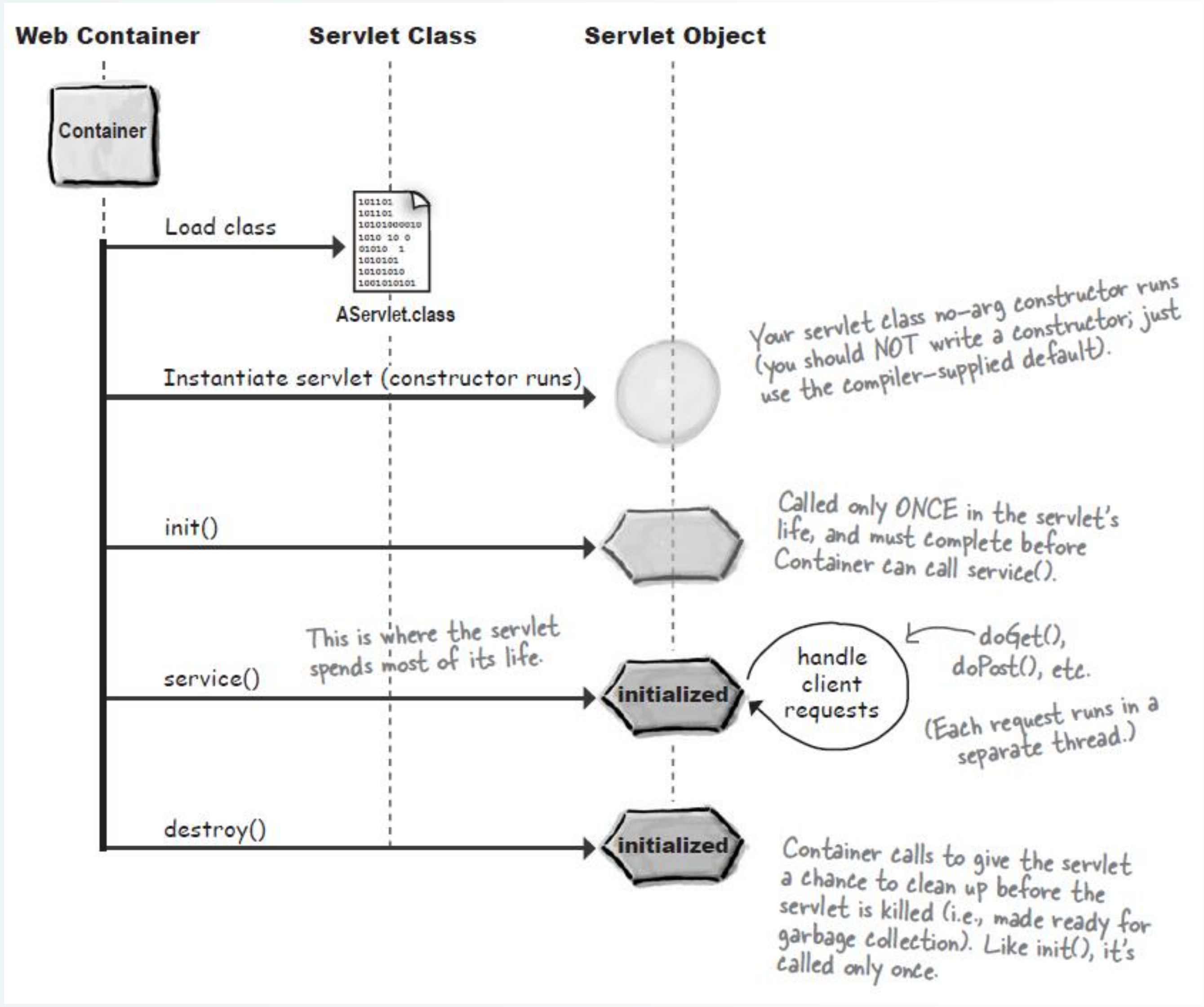
Theory

- init()
  - Executed once when the servlet is first loaded.
  - Not called for each request.
- service()
  - Called in a new thread by the server for each incoming request.
  - Dispatches to doGet, doPost, etc …
  - Do not override this method.
- doGet() / doPost()
  - Handles GET and POST request respectively.
  - Override these to provide desired behavior.
- destroy()
  - Called when server deletes servlet instance.
  - Not called after each request.

The Servlet Life Cycle



The Servlet Life Cycle



Web Container
Servlet Class
Servlet Object
Container
Load class
AServlet.class
Instantiate servlet (constructor runs)
Your servlet class no-arg constructor runs (you should NOT write a constructor; just use the compiler-supplied default).
init()
Called only ONCE in the servlet's life, and must complete before Container can call service().
service()
This is where the servlet spends most of its life.
initialized
handle client requests
doGet(), doPost(), etc.
(Each request runs in a separate thread.)
destroy()
initialized
Container calls to give the servlet a chance to clean up before the servlet is killed (i.e., made ready for garbage collection). Like init(), it's called only once.

The service() method

<u>Never</u> override the service method

- default service() method does other things besides just calling doGet() /doPost() …
  - default service() method parses HTTP request (GET, POST, PUT …) and calls the appropriate method tp handle the incoming request (ex. doGet(), doPost(), doPut() …)
  - If you override the service() method and **<u>do not</u>** handle the HTTP incoming requests correctly, your chances of errors occurring and incoming request not be handles increases.
  - Also, the default service() method give automatic support for:
    - HEAD requests
    - OPTIONS requests
    - TRACE requests

Debugging Servlets

Background

- Use print statements; run server on desktop
- Use Apache Log4j
- Integrated debugger in Eclipse IDE
  - Set breakpoints
  - R-click Tomcat and use “Debug” mode when running server instead of “start”