



# Servlet Basics

Originals of Slides and Source Code for Examples:  
<http://courses.coreservlets.com/Course-Materials/csajsp2.html>

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Agenda

- **The basic structure of servlets**
- **A simple servlet that generates plain text**
- **A servlet that generates HTML**
- **Using helper classes**
- **Giving URLs to servlets**
  - @WebServlet annotation
  - web.xml file
- **The servlet life cycle**
- **Servlet debugging strategies**



# Overview

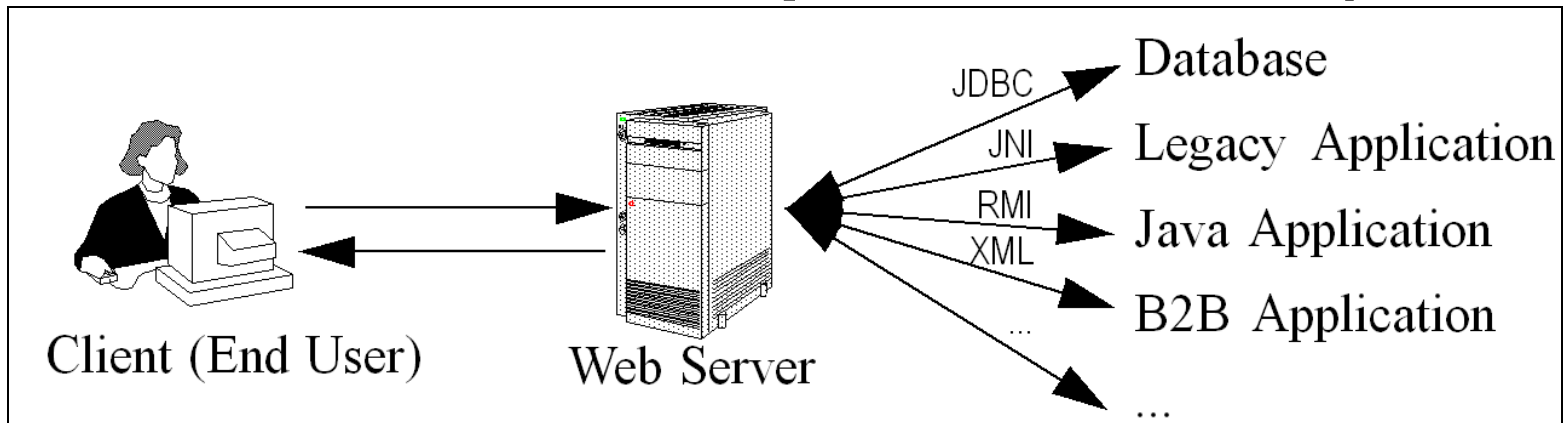
**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# A Servlet's Job

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





# Simple Servlets

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

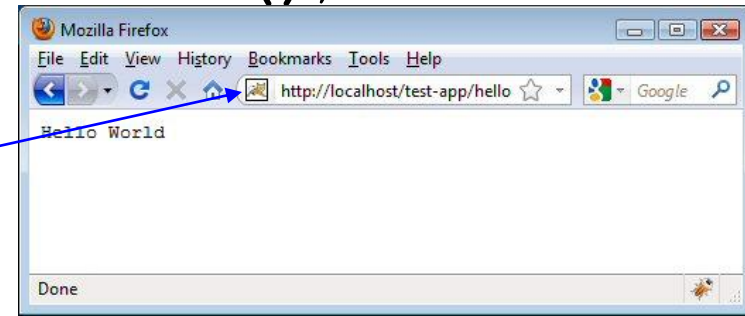


# A Servlet That Generates Plain Text (HelloWorld.java)

```
package testPackage; // Always use packages.
import java.io.*;
import javax.servlet.*;
import javax.servlet.annotation.*;
import javax.servlet.http.*;

@WebServlet("/hello")
public class HelloWorld extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("Hello World");
    }
}
```

URL assumes you have deployed from a project named "test-app". Code can be downloaded from Web site. General form is <http://hostName/appName/address-from-WebServlet-annotation>. Review previous tutorial section for info on how to deploy the app from Eclipse.



# Interpreting HelloWorld Servlet

- **@WebServlet("/address")**
  - This is the URL *relative to the app name*. More later.
- **doGet**
  - Code for an HTTP GET request. doPost also common.
- **HttpServletRequest**
  - Contains anything that comes *from* the browser
- **HttpServletResponse**
  - Used to send stuff *to* the browser. Most common is getWriter for a PrintWriter that points at browser.
- **@Override**
  - General best practice when overriding inherited methods
    - But, I will omit on many of my PowerPoint slides to conserve space. Downloadable source has @Override.

# A Servlet That Generates HTML

- **Tell the browser that you're sending it HTML**
  - `response.setContentType("text/html");`
- **Modify the `println` statements to build a legal Web page**
  - Print statements should output HTML tags
- **Check your HTML with a formal syntax validator**
  - <http://validator.w3.org/>
  - <http://www.htmlhelp.com/tools/validator/>



# HTML 5 Document Format

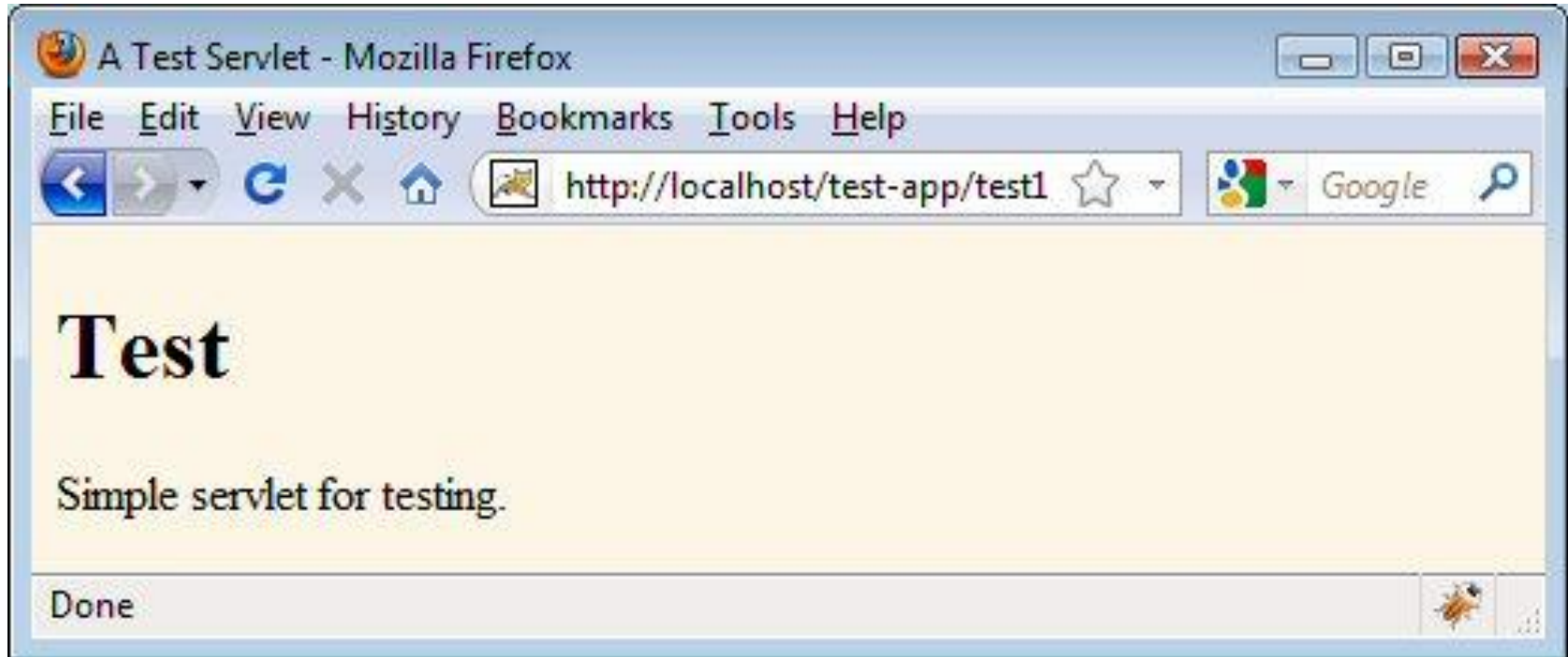
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8"/>
<link href="css/some-stylesheet.css"
      rel="stylesheet"/>
<script src="scripts/some-script.js"></script>
</head>
<body>
...
</body>
</html>
```

Note the simple DOCTYPE, simplified meta tag, and omission of "type" in both the style sheet and script references. All of those work in old, pre-HTML5 browsers.

# A Servlet That Generates HTML (Code)

```
@WebServlet("/test1")
public class TestServlet extends HttpServlet {
    public 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>");
    }
}
```

# A Servlet That Generates HTML (Result)



Assumes project is named test-app.

Eclipse users can use the TestServlet code as a basis for their own servlets.  
Avoid using "New → Servlet" in Eclipse since it results in ugly code.



# Using Helper Classes

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Idea

- **All Java code goes in the same place**
  - In Eclipse, it is `src/packageName`
    - It does not matter if code is for a servlet, helper class, filter, bean, custom tag class, or anything else
- **Don't forget OOP principles**
  - If you find you are doing the same logic multiple times, put the logic in a helper class and reuse it
- **Simple example here**
  - Generates HTML. Building HTML from a helper class is probably not really worth it for real projects (JSP is better), but we haven't covered logic in servlets yet. But the general principle still holds: if you are doing the same thing in several servlets, move the code into shared class.

# A Simple HTML-Building Utility

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

- **Don't go overboard**

- Complete HTML generation packages usually work poorly
  - The JSP framework is a better solution
- More important is to avoid repeating logic. ServletUtilities has a few methods for that, as will be seen later

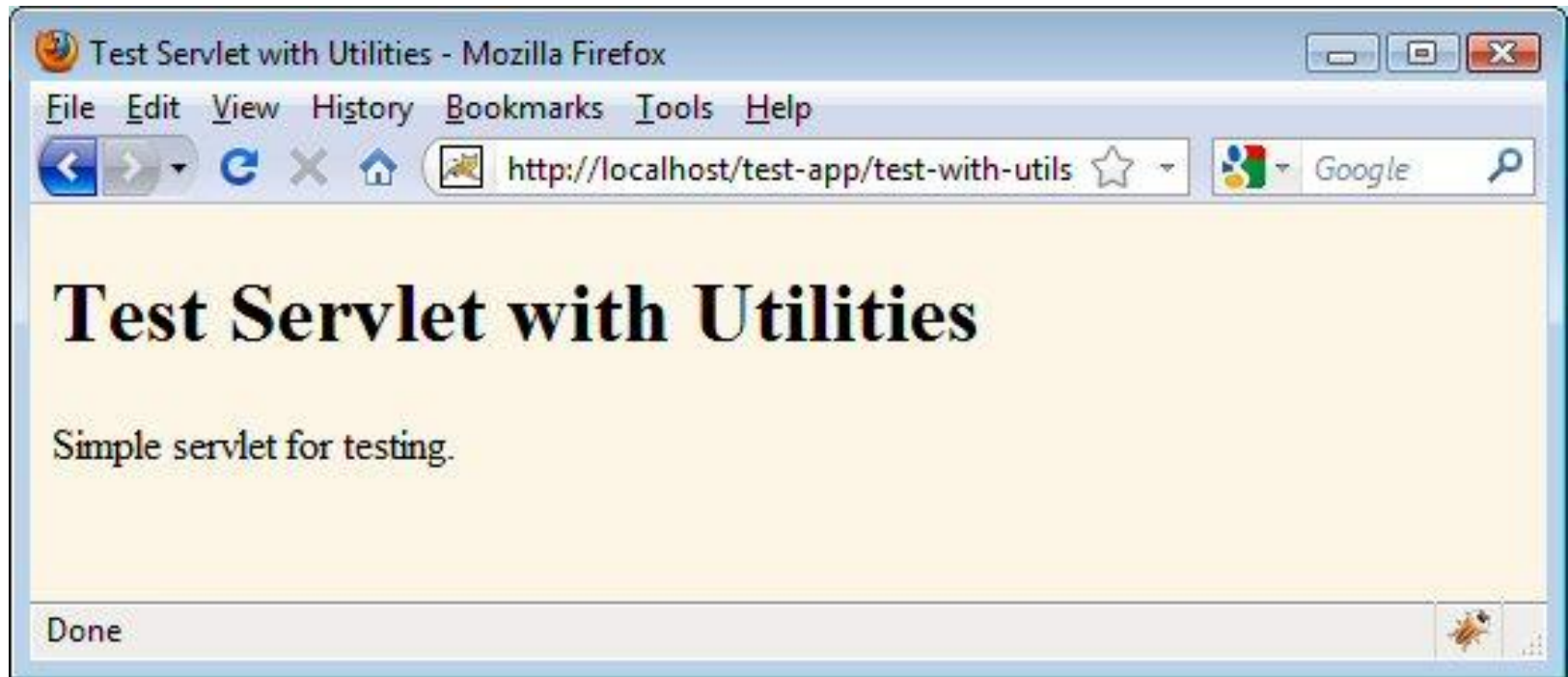


# TestServlet2

...

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

# TestServlet2: Result



Assumes project is named test-app.



# Custom URLs and web.xml

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Tomcat 7 or Other Servlet 3.0 Containers

- **Give address with @WebServlet**

```
@WebServlet("/my-address")
```

```
public class MyServlet extends HttpServlet { ... }
```

- Resulting URL

- `http://hostName/appName/my-address`

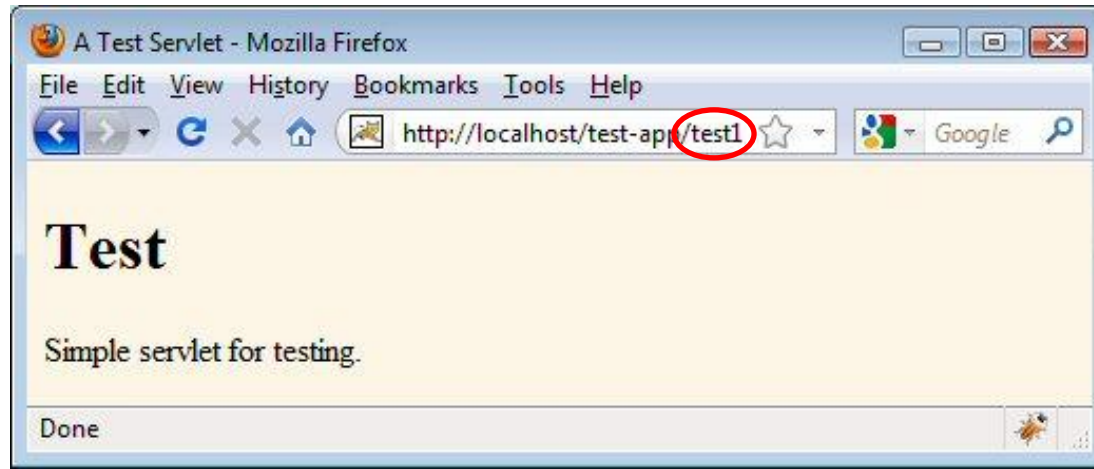
- **Omit web.xml entirely**

- You are permitted to use web.xml even when using @WebServlet, but the entire file is completely optional.

- In earlier versions, you must have a web.xml file even if there were no tags other than the main start and end tags (<web-app ...> and </web-app>).

# Example: URLs with @WebServlet

```
package testPackage;  
...  
@WebServlet("/test1")  
public class TestServlet extends HttpServlet {  
    public void doGet(HttpServletRequest request,  
                      HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        out.println  
            ("<!DOCTYPE html>\n" +  
             ...);  
    }  
}
```



# Defining Custom URLs in web.xml (Servlets 2.5 & Earlier)

- **Java code**

```
package myPackage; ...
public class MyServlet extends HttpServlet { ... }
```

- **web.xml entry (in <web-app...>...</web-app>)**

- Give name to servlet

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

- Give address (URL mapping) to servlet

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

- **Resultant URL**

- `http://hostname/appName/my-address`



# Defining Custom URLs: Example

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
  ... >
```

Don't edit this manually.  
Should match version supported  
by your server. If your server  
supports 3.0, can omit web.xml  
totally and use annotations.

```
<!-- Use the URL http://hostName/appName/test2 for
  testPackage.TestServlet -->
```

```
<servlet>
  <servlet-name>Test</servlet-name>
  <servlet-class>testPackage.TestServlet</servlet-class>
</servlet>
```

Fully qualified classname.

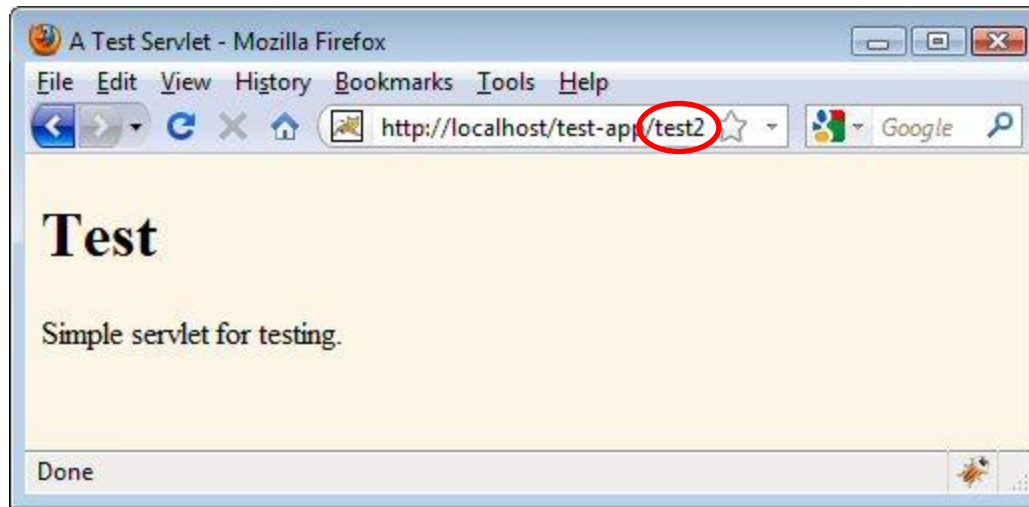
```
<servlet-mapping>
  <servlet-name>Test</servlet-name>
  <url-pattern>/test2</url-pattern>
</servlet-mapping>
```

Any arbitrary name.  
But must be the same both times.

```
</web-app>
```

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

# Defining Custom URLs: Result



- **Eclipse details**

- Name of Eclipse project is “test-app”
- Servlet is in src/testPackage/TestServlet.java
- Deployed by right-clicking on Tomcat, Add and Remove Projects, Add, choosing test-app project, Finish, right-clicking again, Start (or Restart)



# Advanced Topics

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# The Servlet Life Cycle

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

# Why You Should Not Override service

- **The service method does other things besides just calling doGet**
  - You can add support for other services later by adding doPut, doTrace, etc.
  - You can add support for modification dates by adding a getLastModified method
  - The service method gives you automatic support for:
    - HEAD requests
    - OPTIONS requests
    - TRACE requests
- **Alternative: have doPost call doGet**

# Debugging Servlets

- **Use print statements; run server on desktop**
- **Use Apache Log4J**
- **Integrated debugger in IDE**
  - Right-click in left margin in source to set breakpoint (Eclipse)
  - R-click Tomcat and use “Debug” instead of “Start”
- **Look at the HTML source**
- **Return error pages to the client**
  - Plan ahead for missing or malformed data
- **Use the log file**
  - `log("message")` or `log("message", Throwable)`
- **Separate the request and response data.**
  - Request: see EchoServer at [www.coreservlets.com](http://www.coreservlets.com)
  - Response: see WebClient at [www.coreservlets.com](http://www.coreservlets.com)
- **Make sure browser is not caching**
  - Internet Explorer: use Shift-RELOAD
  - Firefox: use Control-RELOAD
- **Stop and restart the server**





# Wrap-Up

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Summary

- **Main servlet code goes in doGet or doPost:**
  - The `HttpServletRequest` contains the incoming information
  - The `HttpServletResponse` lets you set outgoing information
    - Call `setContentType` to specify MIME type
    - Call `getWriter` to obtain a `Writer` pointing to client (browser)
    - Make sure output is legal HTML
- **Give address with `@WebServlet` or `web.xml`**
  - `@WebServlet("/some-address")`
  - `public class SomeServlet extends HttpServlet { ... }`
  - Resulting URL
    - `http://hostName/appName/some-address`



# Questions?

JSF 2, PrimeFaces, Java 7, Ajax, jQuery, Hadoop, RESTful Web Services, Android, Spring, Hibernate, Servlets, JSP, GWT, and other Java EE training

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.