Chapter 5

How to develop servlets

Objectives

Applied

1. Code and test servlets that require the features presented in this chapter.
2. Use the web.xml file or an annotation to map a servlet to a URL pattern.
3. Provide for server-side data validation in your applications.
4. Use the web.xml file to set initialization parameters. Use servlets to get the parameters.
5. Use the web.xml file to implement custom error handling.
6. Write debugging data for a servlet to either the console or a log file.

Objectives (continued)

Knowledge

1. Describe servlets and servlet mapping, and their use of request and response objects.
2. Describe how parameters are passed to a servlet with the HTTP GET method.
3. List three reasons for using the HTTP POST method instead of the HTTP GET method.
4. Describe how the ServletContext object is used to get the path for a file. Describe the use of the init, doGet, doPost, and destroy methods in a servlet.
5. Explain why you should never use instance variables in servlets.
6. Describe the use of debugging data written to the console or log file.

A servlet that returns HTML

```
package murach.email;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TestServlet extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        try {
            out.println("<h1>HTML from servlet</h1>");
        } finally {
            out.close();
        }
    }
```

A servlet that returns HTML (continued)

```
    @Override
    protected void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {

        doPost(request, response);
    }
}
```

Servlet concepts

- The doGet method processes all HTTP requests that use the GET method.
- The doPost method processes all HTTP requests that use the POST method.
- The doGet and doPost methods both accept (1) the HttpServletRequest object, or the *request object*, and (2) the HttpServletResponse object, or the *response object*.

XML tags that add servlet mapping to web.xml file

```
<!-- the definitions for the servlets -->
<servlet>
    <servlet-name>EmailListServlet</servlet-name>
    <servlet-class>murach.email.EmailListServlet</servlet-class>
</servlet>
<servlet>
    <servlet-name>TestServlet</servlet-name>
    <servlet-class>murach.email.TestServlet</servlet-class>
</servlet>

<!-- the mapping for the servlets -->
<servlet-mapping>
    <servlet-name>EmailListServlet</servlet-name>
    <url-pattern>/emailList</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>EmailListServlet</servlet-name>
    <url-pattern>/email/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>TestServlet</servlet-name>
    <url-pattern>/test</url-pattern>
</servlet-mapping>
```

XML elements for working with servlet mapping

Element	Description
`<servlet-class>`	Specifies the class for the servlet.
`<servlet-name>`	Specifies a unique name for servlet used to identify the servlet within the web.xml file.
`<url-pattern>`	Specifies URL or URLs mapped to specified servlet.

Some URL pattern examples

URL pattern	Description
`/emailList`	Specifies the emailList URL in the root directory.
`/email/*`	Specifies any URL in the email directory.
`/email/add`	Specifies the add URL in the email directory.

An annotation that maps a servlet to a URL

```
package murach.email;

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;

@WebServlet("/test")
public class TestServlet extends HttpServlet {
...
...
```

How to use servlet annotations

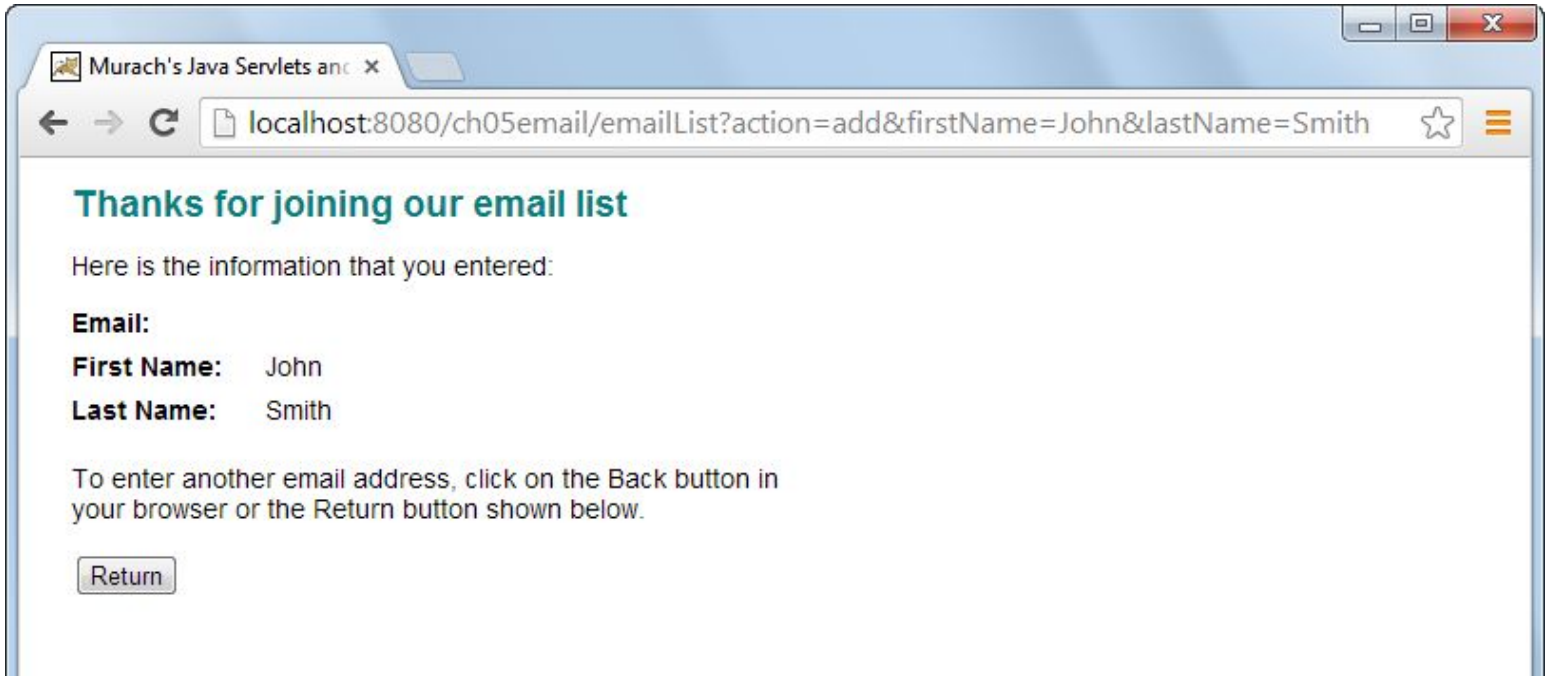
To map a servlet to multiple URLs

```
@WebServlet(urlPatterns={"/emailList", "/email/*"})
```

To specify an internal name for the servlet

```
@WebServlet(name="MurachTestServlet", urlPatterns={"/test"})
```

An HTTP request that uses the GET method



How to append parameters to a URL

```
emailList?action=add
emailList?firstName=John&lastName=Smith
```

How to append parameters to a GET request

1: Enter the URL into the browser's address bar

```
http://localhost:8080/ch05email/emailList?action=add&firstName=John
http://www.murach.com/email/list?action=add&firstName=John
```

2: Code a form that uses the GET method

```
<form action="emailList">
<form action="emailList" method="get">
```

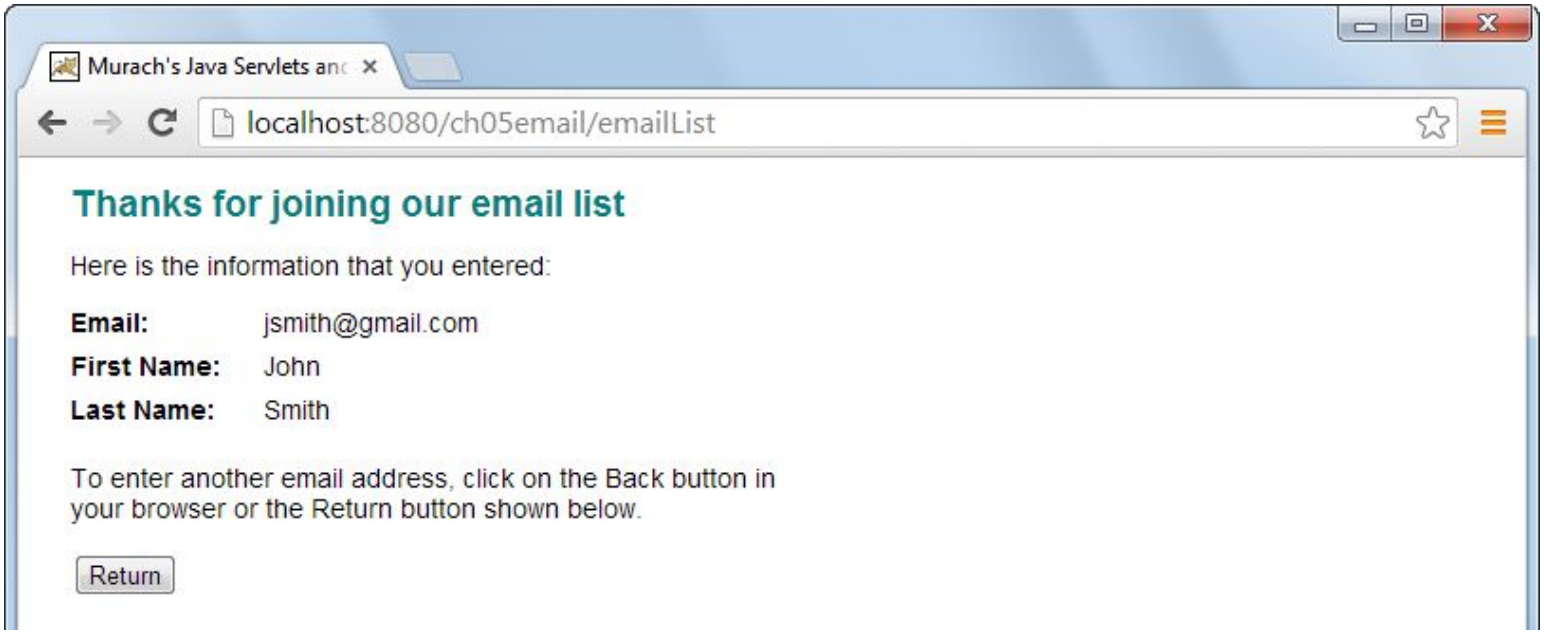
3: Code an anchor tag

```
<a href="emailList?action=join">Display Email Entry Test</a>
```

Note

- A servlet must implement the doGet method to process an HTTP GET request.

A URL requested with the HTTP POST method



A Form tag that uses the POST method

```
<form action="emailList" method="post">
```

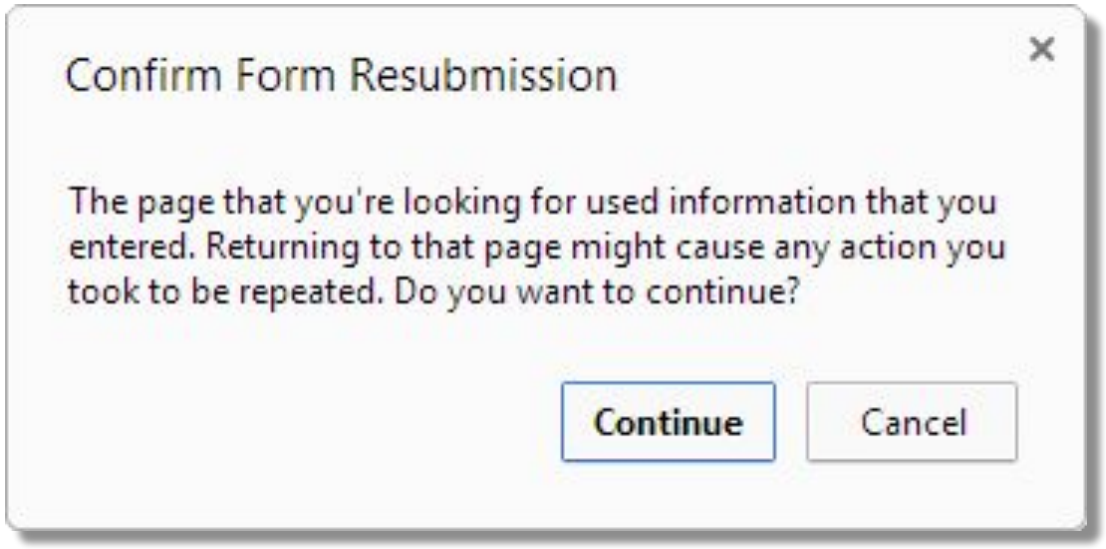
Use the GET method when…

- Request reads data from the server.
- Request can be executed multiple times without causing problems.

Use the POST method when…

- Request writes data to the server.
- Executing request multiple times may cause problems.
- You don't want to include the parameters in the URL for security reasons.
- You don't want users to be able to include parameters when they bookmark a page.
- You need to transfer more than 4 KB of data.

A typical browser dialog that's displayed if the user tries to refresh a post



Two methods of the HttpServletRequest object

Method	Description
`getParameter(String param)`	Returns the value of the specified parameter as a string if it exists or null if it doesn't. Often, this is the value defined in the value attribute of the control in the HTML or JSP file.
`getParameterValues(String param)`	Returns an array of String objects containing all values that the given request parameter has or null if the parameter doesn't have any values.

Servlet code that gets text from a text box

```
String firstName = request.getParameter("firstName");
```

Servlet code that determines if a box is checked

```
// returns the value or "on" if checked, null otherwise.
String rockCheckBox = request.getParameter("rock");
if (rockCheckBox != null)
{
    // rock music was checked
}
```

Servlet code that reads and processes multiple values from a list box

```
// returns the values of the items selected in a list box.
String[] selectedCountries = request.getParameterValues("country");
for (String country : selectedCountries)
{
    // code that processes each country
}
```

A method of the GenericServlet class

Method	Description
`getServletContext()`	Returns a ServletContext object that contains information about the application's context.

A method of the ServletContext class

Method	Description
`getRealPath(String path)`	Returns a String object for the absolute path of the specified relative path.

Code that gets the absolute path for a file

```
ServletContext sc = this.getServletContext();
String path = sc.getRealPath("/WEB-INF/EmailList.txt");
```

A more concise way to write the same code

```
String path = this.getServletContext()
                  .getRealPath("/WEB-INF/EmailList.txt");
```

A possible value for the real path variable

```
C:\murach\servlet_and_jsp\netbeans\book_apps\ch05email\
build\web\WEB-INF\EmailList.txt
```

How to work with the ServletContext object

- Because servlets inherit the GenericServlet class, the getServletContext method is available to all servlets.
- You can use the ServletContext object to read global initialization parameters, to work with global variables, and to write data to log files.

Two methods of the HttpServletRequest object

Method	Description
`setAttribute(String name, Object o)`	Stores any object in the request as an attribute and specifies a name for the attribute. Attributes reset between requests.
`getAttribute(String name)`	Returns value of the specified attribute as an Object type. If no attribute exists for specified name, method returns a null value.

How to set a request attribute

```
User user = new User(firstName, lastName, email);
request.setAttribute("user", user);
```

How to get a request attribute

```
User user = (User) request.getAttribute("user");
```

How to set a request attribute for a primitive type

```
int id = 1;
request.setAttribute("id", new Integer(id));
```

How to get a request attribute for a primitive type

```
int id = (Integer) request.getAttribute("id");
```

A method of the ServletContext object

Method	Description
`getRequestDispatcher(String path)`	Returns a RequestDispatcher object for specified path.

A method of the RequestDispatcher object

Method	Description
`forward(ServletRequest request, ServletResponse response)`	Forwards the request and response objects to another resource on the server (usually a JSP or servlet).

How to forward the request to an HTML page

```
String url = "/index.html";
getServletContext().getRequestDispatcher(url)
    .forward(request, response);
```

How to forward the request to a JSP

```
String url = "/thanks.jsp";
getServletContext().getRequestDispatcher(url)
    .forward(request, response);
```

How to forward the request to a servlet

```
String url = "/cart/displayInvoice";
getServletContext().getRequestDispatcher(url)
     .forward(request, response);
```

A method of the HttpServletResponse class

Method	Description
`sendRedirect(String path)`	Sends a temporary redirect response to the client using the specified redirect location URL.

How to redirect a response…

…relative to the current directory

```
response.sendRedirect("email");
```

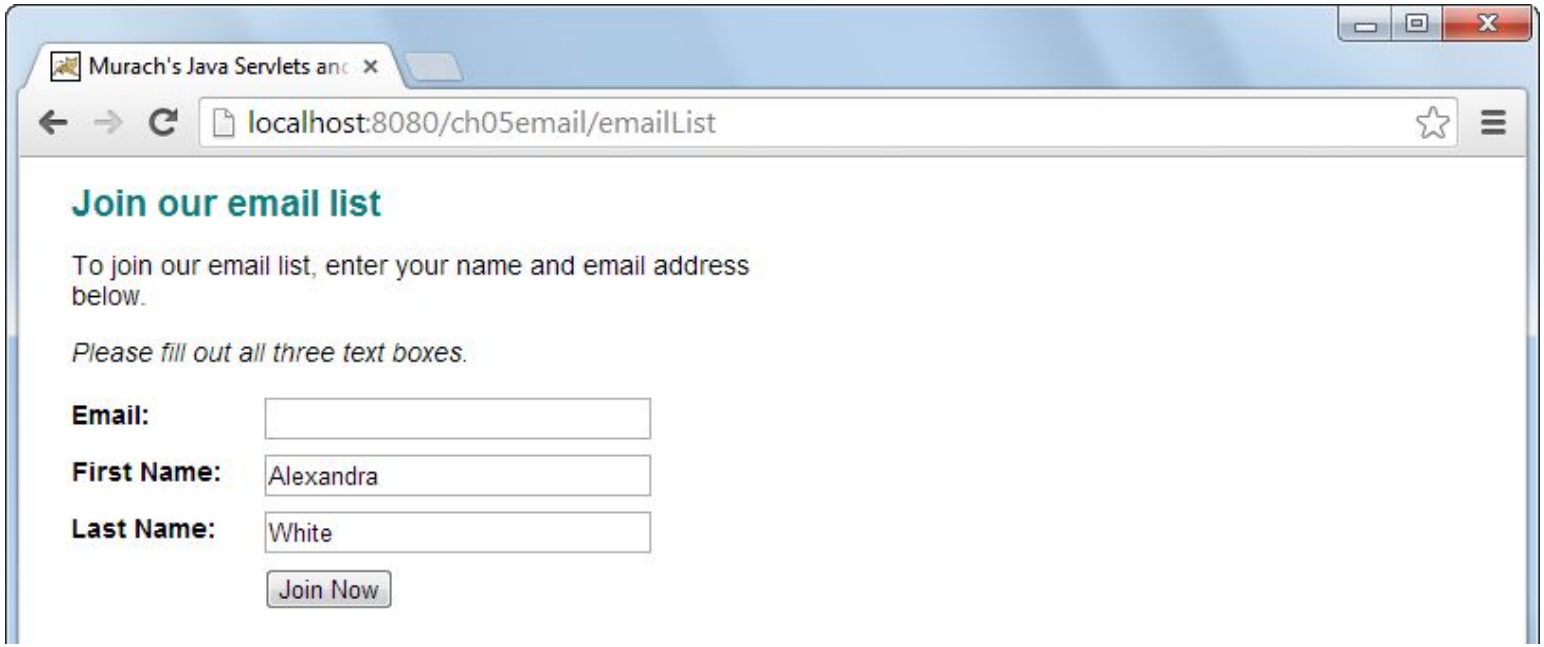
…relative to the servlet engine

```
response.sendRedirect("/musicStore/email/");
```

…to a different web server

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

The JSP displayed when an entry isn't made



The code for the JSP

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Murach's Java Servlets and JSP</title>
    <link rel="stylesheet" href="styles/main.css" type="text/css"/>
</head>
```

The code for the JSP (continued)

```
<body>
    <h1>Join our email list</h1>
    <p>To join our email list, enter your name and
       email address below.</p>
    <p><i>${message}</i></p>
    <form action="emailList" method="post">
        <input type="hidden" name="action" value="add">
        <label class="pad_top">Email:</label>
        <input type="email" name="email" value="${user.email}"><br>
        <label class="pad_top">First Name:</label>
        <input type="text" name="firstName"
               value="${user.firstName}"><br>
        <label class="pad_top">Last Name:</label>
        <input type="text" name="lastName"
               value="${user.lastName}"><br>
        <label> </label>
        <input type="submit" value="Join Now" class="margin_left">
    </form>
</body>
</html>
```

A doPost method that validates data

```
@Override
protected void doPost(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException {

    String url = "/index.jsp";

    // get current action
    String action = request.getParameter("action");
    if (action == null) {
        action = "join";  // default action
    }

    // perform action and set URL to appropriate page
    if (action.equals("join")) {
        url = "/index.jsp";    // the "join" page
    }
    else if (action.equals("add")) {
        // get parameters from the request
        String firstName = request.getParameter("firstName");
        String lastName = request.getParameter("lastName");
        String email = request.getParameter("email");
```

A doPost method that validates data (continued)

```
        // store data in User object
        User user = new User(firstName, lastName, email);

        // validate the parameters
        String message;
        if (firstName == null || lastName == null ||
            email == null ||
            firstName.isEmpty() || lastName.isEmpty() ||
            email.isEmpty()) {

            message = "Please fill out all three text boxes.";
            url = "/index.jsp";
        } else {
            message = "";
            url = "/thanks.jsp";
            UserDB.insert(user);
        }
        request.setAttribute("user", user);
        request.setAttribute("message", message);
    }
    getServletContext()
            .getRequestDispatcher(url)
            .forward(request, response);
}
```

A complete web.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
             http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">

    <context-param>
        <param-name>custServEmail</param-name>
        <param-value>custserv@murach.com</param-value>
    </context-param>

    <servlet>
        <servlet-name>EmailListServlet</servlet-name>
        <servlet-class>murach.email.EmailListServlet</servlet-class>
        <init-param>
            <param-name>relativePathToFile</param-name>
            <param-value>/WEB-INF/EmailList.txt</param-value>
        </init-param>
    </servlet>

    <servlet-mapping>
        <servlet-name>EmailListServlet</servlet-name>
        <url-pattern>/emailList</url-pattern>
    </servlet-mapping>
```

A complete web.xml file (continued)

```
    <!-- you can comment out these error tags
         when the app is in development -->
    <error-page>
        <error-code>404</error-code>
        <location>/error_404.jsp</location>
    </error-page>
    <error-page>
        <exception-type>java.lang.Throwable</exception-type>
        <location>/error_java.jsp</location>
    </error-page>

    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>

    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

Concepts about the web.xml file

- The web.xml file is stored in the WEB-INF directory. When Tomcat starts, it reads the web.xml file.
- If elements in the web.xml aren't nested correctly, Tomcat displays an error message when it reads the web.xml file.
- After you modify the web.xml file, redeploy the application so the changes take effect. Or, restart Tomcat.

Initialization parameters in a web.xml file

```
<context-param>
    <param-name>custServEmail</param-name>
    <param-value>custserv@murach.com</param-value>
</context-param>

<servlet>
    <servlet-name>AddToEmailListServlet</servlet-name>
    <servlet-class>email.AddToEmailListServlet</servlet-class>
    <init-param>
        <param-name>relativePathToFile</param-name>
        <param-value>/WEB-INF/EmailList.txt</param-value>
    </init-param>
</servlet>
```

XML elements for initialization parameters

Element	Description
`<context-param>`	Defines a parameter that's available to all servlets within an application.
`<servlet>`	Identifies a specific servlet within the application.
`<servlet-name>`	Defines the name for the servlet that's used in the rest of the web.xml file.
`<servlet-class>`	Identifies the servlet by specifying the servlet's package and class name.
`<init-param>`	Defines a name/value pair for an initialization parameter for a servlet.
`<param-name>`	Defines the name of a parameter.
`<param-value>`	Defines the value of a parameter.

An annotation that sets initialization parameters for a servlet

```
@WebServlet(urlPatterns={"/emailList"},
            initParams={@InitParam(name="relativePathToFile",
                                   value="/WEB-INF/EmailList.txt")})
```

Initialization parameters

- To create an *initialization parameter* available to all servlets (called a *context initialization parameter*), code the param-name and param-value elements within the context-param element.
- To create an initialization parameter available to a specific servlet (called a *servlet initialization parameter*), code the param-name and param-value elements within the init-param element. But first, identify the servlet by coding the servlet, servlet-name, and servlet-class elements.

Two methods of the GenericServlet class

Method	Description
`getServletContext()`	Returns a ServletContext object that contains information about the entire web application's context.
`getServletConfig()`	Returns a ServletConfig object that contains information about a single servlet's configuration.

A method of the ServletContext and ServletConfig interfaces

Method	Description
`getInitParameter(` `  String name)`	Returns a String object that contains the value of the specified initialization parameter. If the parameter doesn't exist, this method returns a null value.

Code that gets an initialization parameter that's…

…available to all servlets

```
String custServEmail = this.getServletContext()
                           .getInitParameter("custServEmail");
```

…available to the current servlet only

```
String relativePath = this.getServletConfig()
                          .getInitParameter("relativePathToFile");
```

XML elements for working with error handling

Element	Description
`<error-page>`	Specifies an HTML page or JSP displayed when the application encounters an uncaught exception or a certain type of HTTP status code.
`<error-code>`	Specifies the number of a valid HTTP status code.
`<exception-type>`	Uses the fully qualified class name to specify a Java exception.
`<location>`	Specifies the location of the HTML page or JSP that's displayed.

How to provide error-handling for an HTTP 404 status code

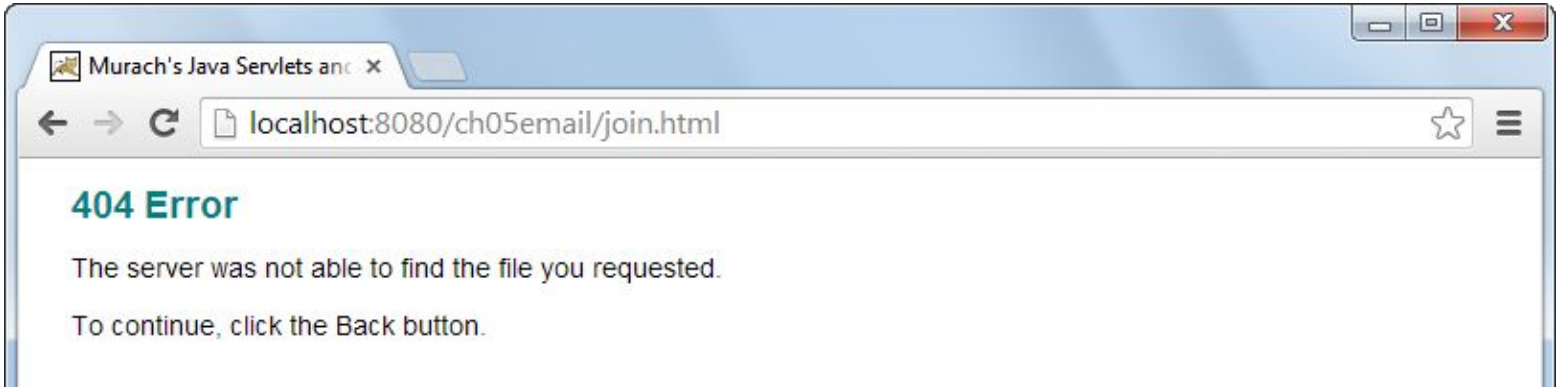
The XML tags

```
<error-page>
    <error-code>404</error-code>
    <location>/error_404.jsp</location>
</error-page>
```

The error_404.jsp file

```
<h1>404 Error</h1>
<p>The server was not able to find the file you requested.</p>
<p>To continue, click the Back button.</p>
```

The JSP page for the 404 error



How to provide error-handling for all Java exceptions

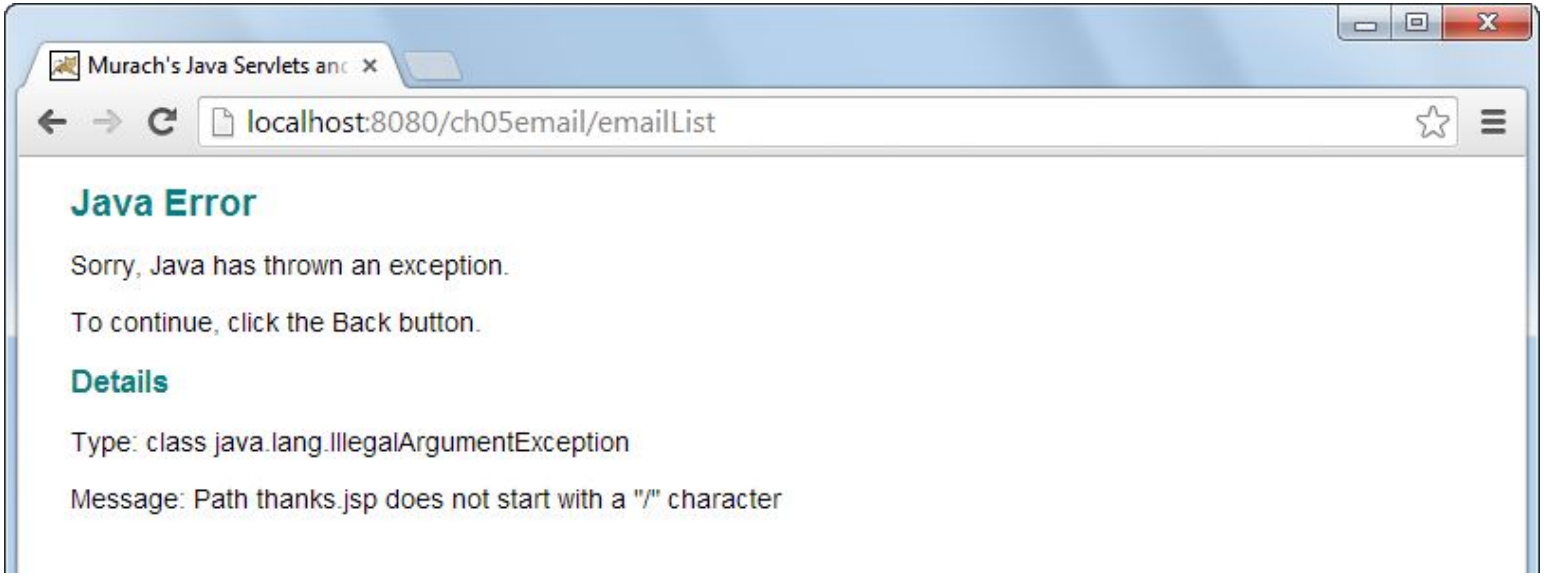
The XML tags

```
<error-page>
    <exception-type>java.lang.Throwable</exception-type>
    <location>/error_java.jsp</location>
</error-page>
```

The error_java.jsp file

```
<h1>Java Error</h1>
<p>Sorry, Java has thrown an exception.</p>
<p>To continue, click the Back button.</p>

<h2>Details</h2>
<p>Type: ${pageContext.exception["class"]}</p>
<p>Message: ${pageContext.exception.message}</p>
```

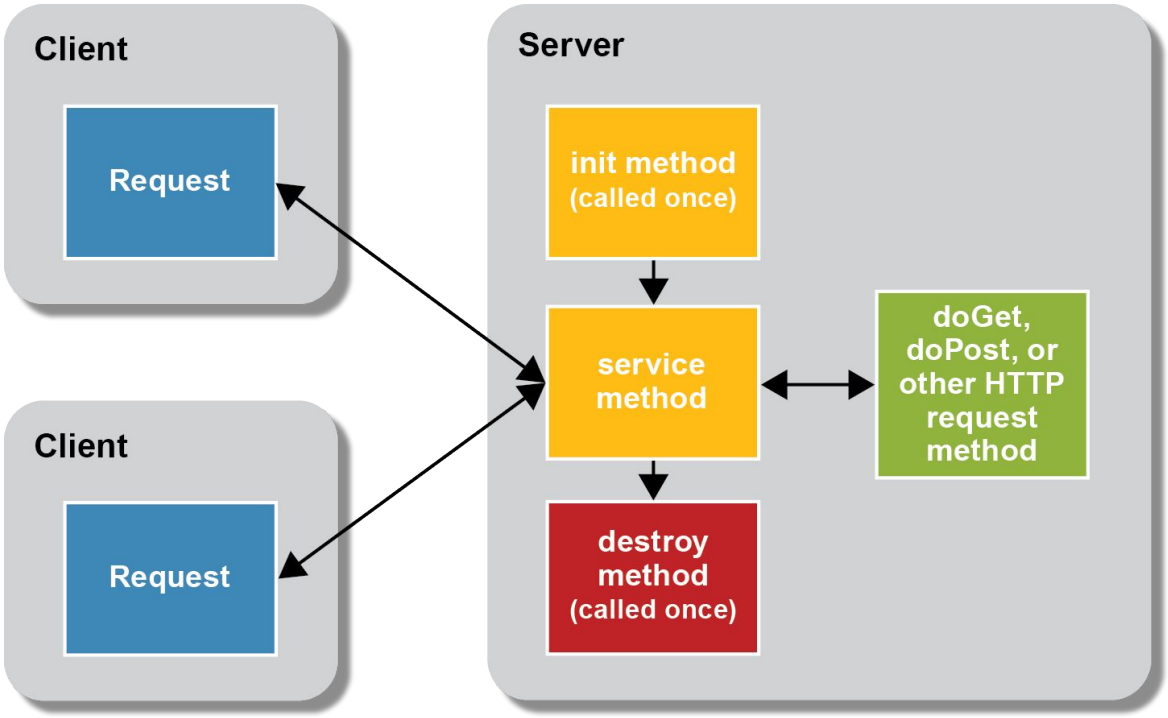
The JSP when a Java exception is thrown



Five common methods of a servlet

```
public void init() throws ServletException

public void service(HttpServletRequest request,
                    HttpServletResponse response)
                    throws IOException, ServletException

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

public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
                   throws IOException, ServletException

public void destroy()
```

The lifecycle of a servlet



Note

- It's generally considered a bad practice to override the service method. Instead, you should override a method like doGet or doPost to handle a specific type of HTTP request.

A servlet with an instance variable

```
package murach.email;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class EmailListServlet extends HttpServlet {

    // declare an instance variable for the page
    private int globalCount; // not thread-safe

    @Override
    public void init() throws ServletException
    {
        globalCount = 0; // initialize the instance variable
    }
```

A servlet with an instance variable (continued)

```
    @Override
    protected void doPost(
        HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        // update global count variable
        globalCount++;   // this is not thread-safe

        // the rest of the code goes here
    }
}
```

Why you shouldn't use instance variables in servlets

- An *instance variable* of a servlet belongs to the one instance of the servlet and is shared by any threads that request the servlet.
- Instance variables are not *thread-safe*. Two threads may conflict when they try to read, modify, and update the same instance variable at the same time, which can result in lost updates or other problems.

Common servlet problems

Problem	Possible solutions
The servlet won't compile	Make sure compiler has access to the JAR files for all necessary APIs.
The servlet won't run	Make sure web server is running.
	Make sure URL is correct.
Changes aren't showing up	Redeploy the application.
	Restart the server so it reloads all applications.
	Make sure servlet reloading is on.
The page doesn't display correctly	Use browser to view the HTML code for the page. Go through the HTML code to identify the problem, and fix the problem in the servlet.

Code that prints debugging data to the console

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
                   throws IOException, ServletException
{
    // code

    String email = request.getParameter("email");
    System.out.println("EmailListServlet email: " + email);

    // more code
}
```

Sample output displayed on the Tomcat console

```
EmailListServlet email: jsmith@gmail.com
```

How to print debugging data to the console

- Use the println method of the System.out or System.err objects to display debugging messages on the console for the servlet engine.
- When you use debugging messages to display variable values, include servlet name and variable name so messages are easy to interpret.
- NetBeans displays a tab for the Tomcat console within its Output window.

Two methods of the HttpServlet class

Method	Description
`log(String message)`	Writes the specified message to the server's log file.
`log(String message, Throwable t)`	Writes the specified message to the server's log file, followed by the stack trace for the exception.

Servlet code that prints a variable to a log file

```
log("email=" + email);
```

Sample data that's written to the log file

```
Jun 29, 2014 6:26:04 PM org.apache.catalina.core.ApplicationContext
log
INFO: EmailListServlet: email=jsmith@gmail.com
```

Servlet code that prints a stack trace to a log file

```
try {
    UserIO.add(user, path);
} catch(IOException e) {
    log("An IOException occurred.", e);
}
```

The data that's written to the log file

```
Jun 29, 2014 6:30:38 PM
org.apache.catalina.core.StandardWrapperValve invoke
WARNING: Servlet.service() for servlet EmailListServlet threw
exception
java.io.FileNotFoundException:
C:\murach\servlet_and_jsp\netbeans\ch05email\build\web\WEB-
INF\EmailList.txt (Access is denied)
        at java.io.FileOutputStream.openAppend(Native Method)
        at
java.io.FileOutputStream.<init>(FileOutputStream.java:177)
        at java.io.FileWriter.<init>(FileWriter.java:90)
        at murach.data.UserIO.add(UserIO.java:11)
        at
murach.email.EmailListServlet.doPost(EmailListServlet.java:38)
...
...
```

The location of a typical Tomcat log file

```
C:\tomcat-8.0\logs\localhost.2014-06-29.log
```

How to print debugging data to a log file

- Use the log methods of the HttpServlet class to write debugging data to web server's *log file*.
- *Stack trace* is the chain of method calls for any statement that calls a method.
- Data that's written by log methods varies from one server to another. The name and location of the log files may also vary from one server to another.
- NetBeans displays a tab for the server log within its Output window.