Building Java HTTP servlets

Section 1. Tutorial tips

What are the prerequisites?

This tutorial teaches basic concepts about servlets: what, why, and how. You should be familiar with Java programming concepts such as the structure of the language (packages, classes, and methods) and how object-oriented programming differs from procedural programming. You don't need prior experience writing servlets.

To write the servlets described in this tutorial, you simply need an editing environment. This can be as basic as an operating system editor. In a development environment, many people use IDEs because they have debuggers and other features specific to writing and testing code.

To compile the servlets, you'll need the Java compiler (`javac.exe`). You'll also need the Java server development kit (JSDK), which is typically an extension to the JDK. Most servers support servlets based on the JSDK Version 2.1. You can test your servlets with a utility included in the JSDK. The JSDK 2.0 has a utility called servletrunner, while JSDK 2.1 comes with a small HTTP server.

To execute the servlets, you'll need an application server, such as WebSphere, or an HTTP server with a servlet engine installed. HTTP servers are usually included with the application servers. Most HTTP servers that are not included in an application server require installation of an additional servlet engine such as Apache JServ.

Navigation

Navigating through the tutorial is easy:

* Select Next and Previous to move forward and backward through the tutorial.
* When you're finished with a section, select the Main Menu for the next section. Within a section, use the Section Menu.
* If you'd like to tell us what you think, or if you have a question for the author about the content of the tutorial, use the Feedback button.

Contact

For questions about the content of this tutorial, contact the author, Jeanne Murray, at *jeannem@us.ibm.com*. If you have problems running the servlets, a direct contact with Barry Busler, at *barrybusler@us.ibm.com*, will get you a faster answer.

Jeanne Murray is on the developerWorks staff, a perch from which she cajoles others to write servlets for tutorials. Thanks to past and present dWers Dan Amerson and Barry Busler for their contributions. Jeanne's been with IBM in Research Triangle Park, NC, for 12 years, where she's worked on software design and development projects ranging from mainframes to miniature devices.

Section 2. Introduction to servlets

What does a servlet do?

A servlet is a server-side software program, written in Java code, that handles messaging between a client and server. The Java Servlet API defines a standard interface for the request and response messages so your servlets can be portable across platforms and across different Web application servers.

Servlets can respond to client requests by dynamically constructing a response that is sent back to the client. For example, in this tutorial you'll write a servlet to respond to an HTTP request.

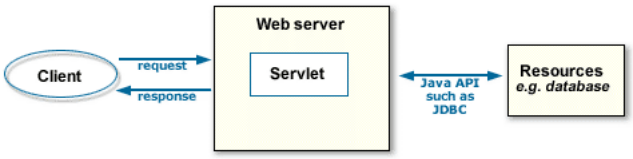
Because servlets are written in the Java programming language, they have access to the full set of Java APIs. This makes them ideal for implementing complex business application logic and especially for accessing data elsewhere in the enterprise. The Java Database Connectivity (JDBC) API, which allows Java programs to access relational databases (and is beyond the scope of this tutorial), is one example. Because there are no graphics associated with servlets, access to the GUI Java APIs (the AWT) is not relevant.

A single servlet can be invoked multiple times to serve requests from multiple clients. A servlet can handle multiple requests concurrently and can synchronize requests. Servlets can forward requests to other servers and servlets.

How does a servlet work?

A servlet runs inside a Java-enabled application server. Application servers are a special kind of Web server; they extend the capabilities of a Web server to handle requests for servlets, enterprise beans, and Web applications. There is a distinct difference between a Web server and an application server. While both can run in the same machine, the Web server runs client code such as applets and the application server runs the servlet code.

The server itself loads, executes, and manages servlets. The server uses a Java bytecode interpreter to run Java programs; this is called the Java Virtual Machine (JVM).



The basic flow is this:

1. The client sends a request to the server.
2. The server instantiates (loads) the servlet and creates a thread for the servlet process. Note the servlet is loaded upon the first request; it stays loaded until the server shuts down.
3. The server sends the request information to the servlet.
4. The servlet builds a response and passes it to the server.
5. The server sends the response back to the client.

The servlet dynamically contructs the response using information from the client request, plus data gathered from other sources if needed. Such sources could be other servlets, shared objects, resource files, and databases. Shared resources, for example, could include in-memory data such as instance or class variables and external objects such as files, database connections, and network connections.

How do you run a servlet?

Servlets are either invoked as an explicit URL reference or are embedded in HTML and invoked from a Web application. You don't execute a Java command to run a servlet (as you would an applet); instead, you issue a URL command pointing to the location of the servlet.

Servlets are located in any directory on a machine where an application server is running. Typically, there is a servlet directory to which you will ftp or copy your class files so the application server can find them. This location can be configured differently, depending on the administrator and the environment.

Servlets are loaded when they are invoked for the first time from a client. Alternatively, they can be loaded when the application server is started; this is a configuration decision.

The server must support the version level at which you've written your Java servlets. Most servers support servlets written based on the Java Servlet Development Kit (JSDK) Version 2.1. Many are starting to support Version 2.2, which is the latest revision of the Java Servlet API Specification (Version 2.3 is under review).

A note about debuggers

Debugging a servlet in run-time mode is trickier than debugging an applet because the servlet is running in a different thread in a different environment. The technique you will use to debug depends on how you developed your code. If you use an IDE, then you will likely use a built-in debugger. You may also choose to use a stand-alone debugger. Debugging using the System.out.println method to display messages on the console is the oldest way to debug a program. However, due to the environment in which servlets run, setting up a debugging environment isn't always easy. The console window isn't used much with servlets because most servlet engines start a JVM without displaying a console window.

The list of resources at the end of this tutorial includes some debuggers that support servlets. You should become familiar with at least one of these debuggers; it can help speed the development process and make life a whole lot easier.

How are servlets different from CGI programs?

Common gateway interface (CGI) programs are also used to construct dynamic Web content in response to a request. But servlets have several advantages over CGI. Servlets provide a component-based, platform-independent method for building Web applications, without the performance limitations of CGI programs. Servlets are:

* **Portable across platforms and across different Web servers.** Servlets enable you to do server-side programming without writing to platform-specific APIs; the Java Servlet API is a standard Java extension.
* **Persistent.** A servlet remains in memory once loaded, which means it can maintain system resources -- such as a database connection -- between requests.
* **Efficient.** When a client makes multiple calls to a servlet, the server creates and loads the servlet only once. Each additional call performs only the business logic processing. CGI processes are loaded with each request, which slows performance. In addition, the JVM uses lightweight Java threads to handle servlet requests, rather than a weighty operating system process used by CGI.
* **Able to separate presentation from business logic.** This makes it easier to split a project into distinct parts for easier development and maintenance.
* **Able to access a library of HTTP-specific calls** and to benefit from the continuing development of the Java language itself.

Section 3. The Java Servlet API

What is the Java Servlet API?

The Java Servlet API is a set of classes that define a standard interface between a Web client and a Web servlet. In essence, the API encases requests as objects so the server can pass them to the servlet; the responses are similarly encapsulated so the server can pass them back to a client.

The Java Servlet API has two packages. `javax.servlet` contains classes to support generic protocol-independent servlets, and `javax.servlet.http` includes specific support for the HTTP protocol.

You may want to bring up the *servlet package documentation (Version 2.2)* in a secondary browser window as you go through this section.

On the Web: HTTP servlets

The `Servlet` interface class is the central abstraction of the Java Servlet API. This class defines the methods that manage the servlet and its communications with clients.

To write an HTTP servlet for use on the Web, use the `HttpServlet` class.

* A client request to a servlet is represented by an `HttpServletRequest` object. This object encapsulates the communication from the client to the server. It can contain information about the client environment and any data to be sent to the servlet from the client.
* The response from the servlet back to the client is represented by an `HttpServletResponse` object. This is often the dynamically generated response, such as an HTML page, and is built with data from the request and from other sources accessed by the servlet.

One example of a simple request-response scenario is to pass a parameter by appending it to a URL. This is a simple way for Web adminstrators to track a session, especially in cases where a client does not accept cookies. This tutorial demonstrates this scenario with our Redirect servlet, described in Section 7. The Redirect servlet takes a target URL and a referring origin as input and redirects the Web browser to the target URL.

Key methods for processing HTTP servlets

Your subclass of HttpServlet must override at least one method. Typically, servlets override the `doGet` or `doPost` method. GET requests are typical browser requests for Web pages, issued when a user types a URL or follows a link. POST requests are generated when a user submits an HTML form that specifies post. The HTTP POST method allows the client to send data of unlimited length to the Web server a single time and is useful when posting information such as credit card numbers. The same servlet can handle both GET and POST by having `doGet` call `doPost`, or vice versa.

Other commonly used methods include:

* `service`, the lowest common denominator method for implementing a servlet; most people use `doGet` or `doPost`
* `doPut`, for HTTP PUT requests
* `doDelete`, for HTTP DELETE requests
* `init` and `destroy`, to manage resources that are held for the life of the servlet
* `getServletInfo`, which the servlet uses to provide information about itself

Section 4. The role of the server

A home for servlets

Servlet code follows the standard API, but the servers in which the code runs vary from free servlet engines to commercial servers. Typically what differentiates the free from the fee servers is the complexity of setup and configuration, compatibility with corollary products, and technical support. If you're just learning servlets, it may be useful to download a free server or servlet engine for your development and testing purposes.

A servlet engine runs within the application server and manages the servlet. It loads and initializes the servlet, passes requests and responses to the servlet and client, manages multiple instances of a servlet, and acts as a request dispatcher. It destroys the servlet at the end of its useful life. Servlets are unloaded and removed from memory when the server shuts down.

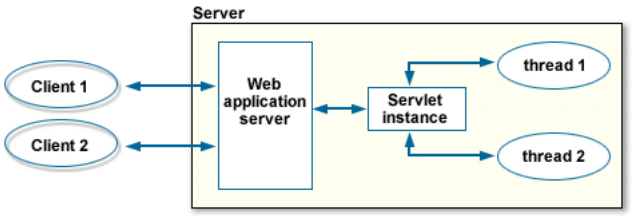
Loading and initialization

A server loads and instantiates a servlet dynamically when its services are first requested. You can also configure the Web server to load and instantiate specific servlets when the Web server is initialized.

The `init` method of the servlet performs the servlet initialization. It is called once for each servlet instance, before any requests are handled. Example tasks performed in the init method include loading default data or database connections.

Handling multithreading

When there are simultaneous requests to a servlet, the server handles each client request by creating a new thread for each request. There is a single thread per client/server request; this thread is used again and again each time the same client makes a request to the servlet.



For example:

* A servlet is loaded by the Web application server. There is one instance of a servlet object at a time, and it remains persistent for the life of the servlet.
* Two browser clients request the services of the servlet. The server creates a handler thread, for each request, against the instance object. Each thread has access to variables that were initialized when the servlet was loaded.
* Each thread handles its own requests. The server sends responses back to the appropriate client.

Request/response handling

Note that the request/response handling done inside the servlet (with the `HttpServletRequest` and `HttpServletResponse` objects) is different from the request handling done by the Web server. The server handles communication with the external entities such as the client by passing the input, then the output, to the servlet.

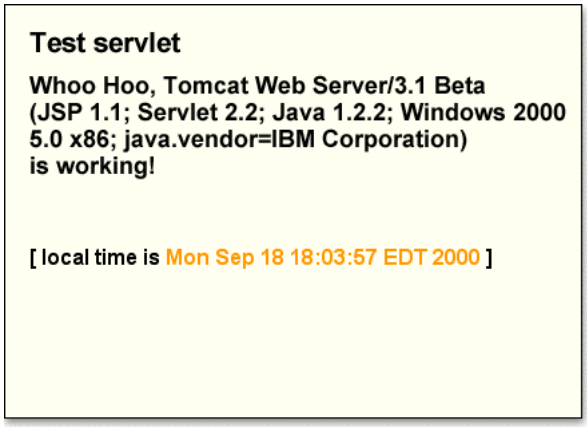
Section 5. A basic HTTP servlet

Scenario: Build a servlet that writes to a browser

In this example, we'll write a very basic servlet to print an HTML page. The servlet calls a standard Java utility to print the date and local time.

This servlet has little real function beyond simply demonstrating the functions of a servlet, but don't worry, we'll get to some more complicated stuff in the next sections.

The image below shows the HTML page this servlet generates. The next few panels walk you through the servlet code.



Add import statements

The import statements give us access to other Java packages. Specifically, the first provides access to standard input output (IO) classes, and the second and third to the Java Servlet API set of classes and interfaces.

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

Declare a new class

The next statement extends the HttpServlet class (`javax.servlet.http.HttpServlet`) to make our new class, called SimplePrint, an HTTP protocol servlet.

We then create a string that names the servlet.

```
public class SimplePrint extends HttpServlet
{
  public static final String TITLE = "Test servlet";
```

Handle HTTP requests

The `service` method handles the HTTP requests; in essence, it allows the servlet to respond to a request. In the next two servlets in this tutorial, we will use the more specific `doGet` and `doPost` methods in place of the `service` method.

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

Set up communication between server and client

Next we set the content type (text/html is the most common), get the communication channel with the requesting client for the response, and get the server identification.

```
response.setContentType("text/html");

PrintWriter out = response.getWriter();

String server = getServletConfig().
                getServletContext().
                getServerInfo();
```

Write the data

Lastly, we write the data. We indicate the server is working and invoke a standard Java utility to display the local time and date, just for fun.

```
   out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML
>"
     + "<HTML>"
     + "<HEAD>"
     + " <TITLE>" + TITLE + "</TITLE>"
     + " <META NAME=\"Author\" CONTENT=\"" + server + "\">"
     + "</HEAD>"
     + "<BODY BGCOLOR=\"#FFFFFF\">"
     + " <CENTER>"
     + "  <H1>" + TITLE + "</H1>"
     + "  <H2>Whoo Hoo, " + server + " is working!</H2>"
     + "  <H3>[ local time is <font color='#FF9900'>"
     + new java.util.Date() + "</font> ]</H3>"
     + " </CENTER>"
     + "</BODY>"
     + "</HTML>");
   }
 }
```

Here's the code

Here's the *SimplePrint.java code* and *Javadoc*.

You can also view the code listing in the *Appendix* of this tutorial.

If you have the ability to execute your servlet on a server, compile the code and put the class file in the servlet directory. Run the servlet using a URL with the following syntax. Be sure to remember that the class file name is case sensitive.

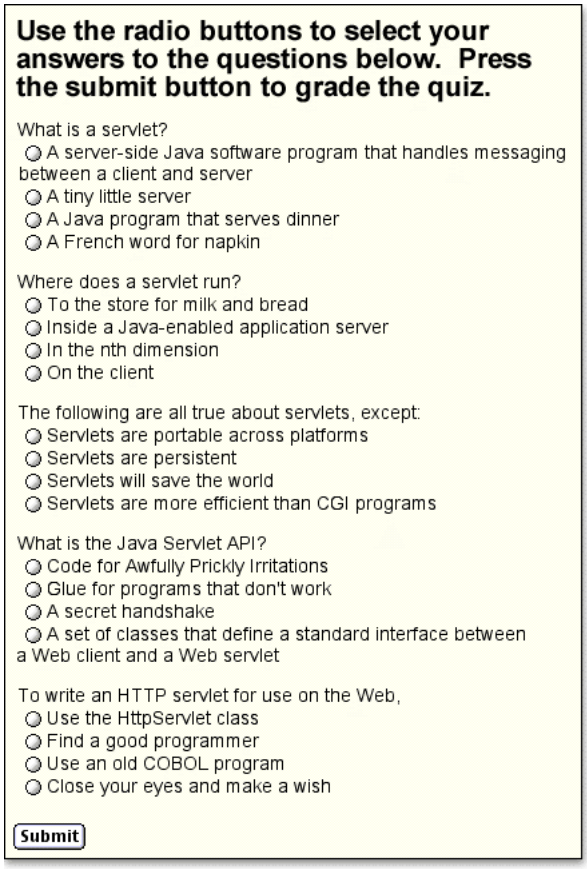
```
http://host/path/SimplePrint
```

Section 6. An HTTP servlet that processes data

Scenario: Build an HTML form that processes data

In this example, we'll write a servlet to generate an HTML form. Our form will quiz you on the concepts covered in this tutorial. Quiz is a simple servlet that handles a GET request submitted from a form in a Web browser. The servlet processes the form data, grades the answers, and displays the results.

The image below shows the HTML page this servlet generates.



Add import statements and class declaration

The next few panels walk you through the servlet code.

First add the import statements that give us access to other Java packages and declare a new class called Quiz.

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

public class Quiz extends HttpServlet
{
```

Add strings for questions and answers

These declarations contain the data for questions and answers. In this example, the code stipulates five questions, each with four possible answers. We also define the set of valid answers to these questions. The userAnswers array will contain the user responses. The submit string is initially set to null, which means user has not yet submitted the form (and we don't yet calculate the grade).

```
  String [] questions =
    {"What is a servlet?",
         "Where does a servlet run?",
         "The following are all true about servlets, except:",
         "What is the Java Servlet API?",
         "To write an HTTP servlet for use on the Web,"};
  String [] [] possibleAnswers =
    {
      { "A server-side Java software program that handles\
messaging between a client and server",
        "A tiny little server",
        "A Java program that serves dinner",
        "A French word for napkin" },

      { "To the store for milk and bread",
        "Inside a Java-enabled application server",
        "In the nth dimension",
        "On the client" },

      { "Servlets are portable across platforms",
        "Servlets are persistent",
        "Servlets will save the world",
        "Servlets are more efficient than CGI programs" },

      { "Code for Awfully Prickly Irritations",
        "Glue for programs that don't work",
        "A secret handshake",
        "A set of classes that define a standard interface between\
a Web client and a Web servlet" },

      { "Use the HttpServlet class",
        "Find a good programmer",
        "Use an old COBOL program",
        "Close your eyes and make a wish" }
    };

  int [] answers = {1,2,3,4,1};
  int [] userAnswers = new int[5];
  String submit = null;
```

Implement the doGet and doPost methods

`doGet` and `doPost` are the servlet's service methods. `doGet` is called by the server to allow a servlet to handle a GET request; `doPost` to handle a POST request. The `service` method receives standard HTTP requests from the public service method and dispatches them to the doXXX methods defined in this class.

The code for the `doGet` method reads the request data and gets the answers submitted by the user. Because the parameters are strings, and we are using ints, we need to call the static parseInt method and catch the appropriate exception. The code also checks to make sure the quiz was submitted.

The `getWriter` method on the response object enables us to send character text to the client.

In this servlet, `doPost` calls `doGet`.This is to ensure that if some program issues a POST request to this servlet (which could alter data), the servlet will enact the `doGet` method in response.

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
  throws ServletException, IOException
{
  PrintWriter out = response.getWriter();
  printHeader(out, response);
  for (int k = 0; k < answers.length; k++)
  {
    try
    {
      userAnswers[k] = (Integer.parseInt
                        (request.getParameter("answer"+k)));
    }
    catch (NumberFormatException nfe)
    {
      userAnswers[k] = 0;
    }
  }
  submit = request.getParameter("submit");
  printQuiz(out);
  printClosing(out);
}
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
  throws ServletException, IOException
{
  this.doGet(request, response);
}
```

Print the HTML form

The rest of the code executes the functions for printing the form. First, print the header. In HTTP, headers must be sent before the response body. And, you need to set the content type before accessing the PrintWriter object. This tells the browser what type of data we're sending.

Next, print the quiz form. This code uses the FORM function to print each question. It also checks to see whether the quiz was submitted (so it knows whether to grade it).

```
public void printHeader(PrintWriter output,
                        HttpServletResponse resp)
{
  resp.setContentType("text/html");
  output.println("<HTML>\n" +
    "<HEAD>\n" +
    "<TITLE>Servlet Quiz</TITLE>\n" +
    "</HEAD>\n" +
    "<BODY BGCOLOR=\"#ffffff\">\n" +
    "<H2>");
}

public void printQuiz(PrintWriter output)
{
  output.println("Use the radio buttons to select your " +
                 "answers to the\nquestions below.  Press " +
                 "the submit button to grade the quiz.\n" +
    "</H2><FORM METHOD=\"GET\" ACTION=\"./Quiz\">");

   if (submit != null)
  {
    grade(output);
  }
```

Print the HTML form (continued)

Then, print each question with its possible answers.

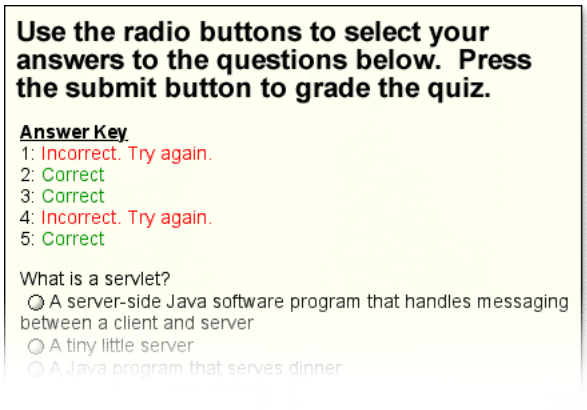
```
    for (int i = 0; i < questions.length; i++)
    {
      printQuestion(output, i);
    }

    output.println("<INPUT TYPE=\"submit\" NAME=\"submit\">\n" +
      "</FORM>");
  }
  public void printClosing(PrintWriter output)
  {
    output.println("</BODY>\n" +
      "</HTML>");
  }

  public void printQuestion(PrintWriter out, int questionIndex)
  {
    out.println("<P>" + questions[questionIndex] + "<BR>\n");
    for (int j = 0; j < possibleAnswers[questionIndex].length; j++)
    {
      out.println("<INPUT TYPE=\"radio\" NAME=\"answer" +
        questionIndex + "\" VALUE=\"" + (j+1) + "\">" +
        possibleAnswers[questionIndex][j] +
        "<BR>");
    }
     out.println("</P>");
  }
```

Grade the quiz

Finally, grade the quiz once it is submitted by the user. The code to grade the quiz loops through the answers and compares them with userAnswers. Correct responses are printed in green; incorrect in red:




```
   public void grade (PrintWriter out)
   {
     out.println("<P><B><U>Answer Key</U></B><BR>");
     for (int p = 0; p < answers.length; p++)
     {
       out.println((p+1) + ":");
       if (answers[p] == userAnswers[p])
       {
         out.println("<FONT
");
       }
       else
       {
         out.println("<FONT COLOR=\"ff0000\">Incorrect. +
                     Try again.</FONT><BR>");
       }
     }
     out.println("</P>");
   }
 }
```

Here's the code

Here's the *Quiz.java code* and *Javadoc*.
You can also view the code listing in the *Appendix* of this tutorial.

If you have the ability to execute your servlet on a server, compile the code and put the class file in the servlet directory. Run the servlet using a URL with the following syntax. Be sure to remember that the class file name is case sensitive.

```
http://host/path/Quiz
```

Section 7. A redirect servlet

Scenario: Tracking use of a link

Let's say we want to track whether people using this tutorial are selecting the link to the Version 2.2 Java servlet documentation at `http://java.sun.com/products/servlet/`. Basically we want our servlet to take as input the referrer URL (that of this tutorial) and the target URL (that of the servlet documentation), then redirect the Web browser to the target URL. Our servlet is called Redirect.java.

This type of servlet is useful for situations in which you want to get information about what your users are doing, but don't have other means for doing so. For example, some clients don't accept cookies, which are a method for storing data on the client and analyzing user actions. The redirect servlet shown in this example does not actually do any tracking (we're not *really* interested in tracking where you want to go today).

Get the housekeeping out of the way

As in the Quiz.java servlet, the first lines of code are the import statements. These are followed by the class declaration for our new class (called Redirect) and the string declarations for referrer and target. Note that these strings are declared as private; that is, only this class can access them.

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

public class Redirect extends HttpServlet
{
  private String referrer;
  private String target;
```

Define the URL processing

The `doGet` method handles the GET requests. It indicates the response will be to redirect the browser to the target. The `sendRedirect` method redirects the browser to the target URL.

And again, as in the Quiz.java servlet, we override `doPost` with `doGet`.

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
  throws ServletException, IOException
{
  try
  {
    getURLs(request);
  }
  catch (Exception e)
  {
    response.sendError(500,"Target parameter not specified");
    return;
  }

  response.sendRedirect(target);
}

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

Get the referrer and target URLs

In this final section, getURLs stores the URL parameters in referrer and target.

The servlet checks whether referrer was specified as a parameter. If not, then it is set to the empty string. For now, referrer is not used, but it could be logged to a file or a database to accumulate session tracking information on a user.

The servlet checks whether a target is specified and, if not, throws the IllegalArgumentException. We could check for more information on the target, such as whether it starts with http, but for now existence is enough of a test.

```
   public void getURLs(HttpServletRequest request)
   {
     referrer = request.getParameter("referrer");
     if (referrer == null || 0 == referrer.length())
     {
       referrer = new String("");
     }
     target = request.getParameter("target");

     // If no target specified, raise an error
     if (target == null || target.equals(""))
     {
       throw new IllegalArgumentException();
     }
   }
 }
```

A note on the referrer URL

In this example, we get an explicit referrer URL as the input. An alternative would be to discern this information from the HTTP environment variable, which is provided by the Web server. Using the actual referrer URL gives you a little more information about where the user is coming from. For example, you could determine whether a user came to your Web site from an ad or from a page link.

And here's the code

Here's the *Redirect.java code* and *Javadoc*.

You can also view the code listing in the *Appendix* of this tutorial.

If you have the ability to execute your servlet on a server, compile the code and put the class file in the servlet directory. Run the servlet using a URL with the following syntax. Be sure to remember that the class file name is case sensitive.

```
http://host/path/Redirect?target=http://java.sun.com/products/servlet/
```

Section 8. Summary

What you now know

By now you should understand what a servlet does and how. This tutorial covered these points from a conceptual point of view and with a bit of practical experience writing simple HTTP servlets. The wonderful world of servlets is much bigger than this, of course, and we hope this tutorial has given you the background for tackling something a little more complex.

The next panel lists some resources for your next steps.

Resources

The Java Servlet API is available on servers through a variety of industry partners and server vendors. The java.sun.com site lists *servers and tools* that support the API.

The *Java servlet package documentation (Version 2.2)* is the definitive source of package, interface, and class definitions.

JavaServer Pages (JSP) technology is an extension of the servlet technology created to support authoring of HTML and XML pages. It makes it easier to combine fixed or static template data with dynamic content. To learn more about JSP technology, take the *JSP technology tutorial* on developerWorks.

Java Database Connectivity (JDBC) technology is how the Java language works with databases. To learn more about JDBC technology, take the *JDBC technology tutorial* on developerWorks.

Developers are finding interesting ways to use servlets with XML. Read more in *Servlets and XML: Made for each other* on developerWorks.

Here are some tools with debuggers and some stand-alone debuggers that support servlets.

IDEs:
* *Borland Inprise JBuilder*
* *IBM VisualAge for Java*
* *SilverStream Designer*
* *Forte for Java*
* *BEA's WebGain for WebLogic*
* *Tek Tool's Kawa*

Visual Debuggers:
* *MetaMata Debug*
* *Jikes Debugger*
* *IDEBUG - IBM Distributed Debugger*

Web Development Environment:
* *IBM WebSphere Studio*

Your feedback

Please let us know whether this tutorial was helpful to you and how we could make it better. We'd also like to hear about other tutorial topics you'd like to see covered. Thanks!

For questions about the content of this tutorial, contact the author, Jeanne Murray, at *jeannem@us.ibm.com*. If you have problems running the servlets, a direct contact with Barry Busler, at *barrybusler@us.ibm.com*, will get you a faster answer.

Section 9. Appendix: Source code listings

SimplePrint.java

Here is the source for `SimplePrint.java`.

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

 /**
  * This servlet is used to print a simple HTML page.
  */

 public class SimplePrint extends HttpServlet
 {
   public static final String TITLE = "Test servlet";

   /**
    * This function handles HTTP requests
    */
   public void service (HttpServletRequest request,
                        HttpServletResponse response)
     throws ServletException, IOException
   {
     // set content type and other response header fields first
     response.setContentType("text/html");

     // get the communication channel with the requesting client
     PrintWriter out = response.getWriter();

     // get the server identification
     String server = getServletConfig().
                     getServletContext().
                     getServerInfo();

     // write the data
     out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML
>"
       + "<HTML>"
       + "<HEAD>"
       + " <TITLE>" + TITLE + "</TITLE>"
       + " <META NAME=\"Author\" CONTENT=\"" + server + "\">"
       + "</HEAD>"
       + "<BODY BGCOLOR=\"#FFFFFF\">"
       + " <CENTER>"
       + "  <H1>" + TITLE + "</H1>"
       + "  <H2>Whoo Hoo, " + server + " is working!</H2>"
       + "  <H3>[ local time is <font color='#FF9900'>"
       + new java.util.Date() + "</font> ]</H3>"
       + " </CENTER>"
       + "</BODY>"
       + "</HTML>");
```

```
  }
}
```

Quiz.java

Here is the source for `Quiz.java`.

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

/**
  * Quiz is a simple servlet that handles a
  * GET request submitted from a form in a web browser.
  * Quiz grades the form results and displays the results
  * for the user.
  */

public class Quiz extends HttpServlet
{
  String [] questions =
      {"What is a servlet?",
         "Where does a servlet run?",
         "The following are all true about servlets, except:",
         "What is the Java Servlet API?",
         "To write an HTTP servlet for use on the Web,"};
  String [] [] possibleAnswers =
  {
    { "A server-side Java software program that handles\
 messaging between a client and server",
      "A tiny little server",
      "A Java program that serves dinner",
      "A French word for napkin" },

    { "To the store for milk and bread",
      "Inside a Java-enabled application server",
      "In the nth dimension",
      "On the client" },

    { "Servlets are portable across platforms",
      "Servlets are persistent",
      "Servlets will save the world",
      "Servlets are more efficient than CGI programs" },

    { "Code for Awfully Prickly Irritations",
      "Glue for programs that don't work",
      "A secret handshake",
      "A set of classes that define a standard interface between\
 a Web client and a Web servlet" },

    { "Use the HttpServlet class",
```

```
    "Find a good programmer",
    "Use an old COBOL program",
    "Close your eyes and make a wish" }
};
int [] answers = {1,2,3,4,1};
int [] userAnswers = new int[5];
String submit = null;

/**
  * This method handles the GET request from the web browser.
  */
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
  throws ServletException, IOException
{
  PrintWriter out = response.getWriter();
  printHeader(out, response);
  for (int k = 0; k < answers.length; k++)
  {
    try
    {
    // Get answers from the form submitted by the user

      userAnswers[k] = (Integer.parseInt(request.
                        getParameter("answer"+k)));
    }
    catch (NumberFormatException nfe)
    {
      userAnswers[k] = 0;
    }
  }
  // Checking to see if quiz was submitted.
  submit = request.getParameter("submit");
  printQuiz(out);
  printClosing(out);
}

/**
  * This method passes any POST requests to the doGet method.
  */
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
  throws ServletException, IOException
{
  this.doGet(request, response);
}

/**
  * Print the top portion of the HTML page.
  * No dynamic content here.
  */
public void printHeader(PrintWriter output,
                        HttpServletResponse resp)
{
  //Tell the browser what type of data we are sending.
```

```
    resp.setContentType("text/html");
    output.println("<HTML>\n" +
    "<HEAD>\n" +
    "<TITLE>Servlet Quiz</TITLE>\n" +
    "</HEAD>\n" +
    "<BODY BGCOLOR=\"#ffffff\">\n" +
    "<H2>");
  }

  /**
    * Prints the form for the Quiz.
    */
  public void printQuiz(PrintWriter output)
  {
    output.println("Use the radio buttons to select " +
                   "your answers to the\nquestions below.  " +
                   "Press the submit button to grade " +
                   "the quiz.\n" +
       "</H2><FORM METHOD=\"GET\" ACTION=\"./Quiz\">");

    // Check to see if a quiz was submitted.
    // If so, grade the quiz.
    if (submit != null)
    {
      grade(output);
    }

    // Print all the questions
    for (int i = 0; i < questions.length; i++)
    {
      printQuestion(output, i);
    }
    output.println("<INPUT TYPE=\"submit\" NAME=\"submit\">\n" +
                   "</FORM>");
  }

  /**
    * Closes the HTML page.
    */
  public void printClosing(PrintWriter output)
  {
    output.println("</BODY>\n" +
                   "</HTML>");
  }

  /**
    * Prints questions from the question list.
    */
  public void printQuestion(PrintWriter out,
                            int questionIndex)
  {
    out.println("<P>" + questions[questionIndex] + "<BR>\n");
    for (int j = 0; j < possibleAnswers[questionIndex].length; j++)
    {
      out.println("<INPUT TYPE=\"radio\" NAME=\"answer" +
```

```
                    questionIndex +
                    "\" VALUE=\"" + (j+1) + "\">" +
                    possibleAnswers[questionIndex][j] +
                    "<BR>");
      }
      out.println("</P>");
    }

    /**
      * Grades the quiz submitted by the user.
      */
    public void grade (PrintWriter out)
    {
      out.println("<P><B><U>Answer Key</U></B><BR>");
      for (int p = 0; p < answers.length; p++)
      {
        out.println((p+1) + ":");
        if (answers[p] == userAnswers[p])
        {
          out.println("<FONT
");
        }
        else
        {
          out.println("<FONT COLOR=\"ff0000\">Incorrect. Try
</FONT><BR>");
        }
      }
    }
 }
```

Redirect.java

Here is the source for `Redirect.java`.

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

 /**
  * Redirect is a servlet that takes a target url and
  * a referring origin as input and redirects the web
  * browser to the target url.
  */

 public class Redirect extends HttpServlet
 {
   private String referrer;
   private String target;

   /**
```

```
    * This method handles GET requests
    */
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
     throws ServletException, IOException
   {
     try
     {
       getURLs(request);
     }
     catch (Exception e)
     {
            // Bad input, send error.
       response.sendError(500,"Target parameter not specified");
       return;
     }
     // Redirect the browser
     response.sendRedirect(target);
   }

   /**
    *  This method passes all POST requests to the doGet() method.
    */

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

   /**
    * Method gets the referrer and target parameters from the request.
    */
   public void getURLs(HttpServletRequest request)
   {
     referrer = request.getParameter("referrer");
     if (referrer == null || 0 == referrer.length())
     {
       referrer = new String("");
     }
     target = request.getParameter("target");

          // If no target specified, raise an error
     if (target == null || target.equals(""))
     {
       throw new IllegalArgumentException();
     }
   }
 }
```

Colophon

This tutorial was written entirely in XML, using the developerWorks tutorial tag set. The tutorial is converted into a number of HTML pages, a zip file, JPEG heading graphics, and a PDF file by a Java program and a set of XSLT stylesheets.