Overview

- Web Applications
- Servlet Model/ Architecture
- Servlet Container and Servlet Life Cycle
- Creating Servlets
  - Interface Servlet and class HttpServlet
  - HTTP methods
- Deploying Servlets
  - Web Application Structure
  - Customizing the Servlet: web.xml
- Using Servlets
  - HTML Forms
  - GET and POST requests
- Handling requests: ServletRequest/HttpServletRequest
- Sending responses: ServletResponse/HttpServletRequest
- Servlet Configuration: ServletConfig
- Servlet Context: ServletContext
Server-side Programming: Java Servlets

- A servlet is a server extension, i.e. a piece of code that extends the functionality of a web server.
  - It resides in a servlet container and generates dynamic content in response to client requests.
  - It is a Java class that implements the javax.servlet.Servlet interface.
- A HTTP servlet is a servlet that implements the javax.servlet.http.HttpServlet subinterface and services HTTP requests.

Servlet Container

- A servlet container (also called a servlet engine) is a separate module used by the web server to load and run servlets.
- We will use Tomcat (5.5.x) (http://jakarta.apache.org/tomcat/) -- which can be run standalone, as a web server and a servlet container integrated into a single program.
- Some other servlet containers: Resin, JRun, Weblogic, WebSphere.

Practical Considerations

- Install Tomcat (5.5.x) (http://jakarta.apache.org/tomcat/).
- Set environment variables (Windows):
  TOMCAT_HOME = C:\jakarta-tomcat-5.5.x
  CATALINA_HOME = C:\jakarta-tomcat-5.5.x
  JAVA_HOME = C:\Program Files\Java\jdk1.5.0_y
  CLASSPATH = .;%TOMCAT_HOME%\common\lib\servlet-api.jar;%TOMCAT_HOME%\common\lib\jsp-api.jar
- Use Tomcat Manager (http://hostname/manager/html).
  - To start, stop and reload.
  - Requires user authentication.
- Reference the Servlet API doc:
  - http://hostname/tomcat-docs/index.html
  - C:\jakarta-tomcat-5.5.x\webapps\tomcat-docs\servletapi\index.html
Active and Passive Resources

- A **passive resource** does not do any processing. It usually has *static content* which is retrieved by the client.

  For example: HTML and image files

  http://www.ii.uib.no/~khalid/index.html

- An **active resource** has processing capabilities. The result of the processing (*dynamic content*) is returned to the client.

  For example: servlets and JSP scripts.

  http://www.google.com/search?sourceid=navclient&ie=UTF-8&q=servlet+jsp

Web Applications and Components

- A **web application** comprises several **web components** which together provide a set of services over the Web.

- A web component can be a passive or an active resource, providing specific task capabilities.
Servlet API Specification

- The Servlet API (Application Programming Interfaces) Specification provides a standard, platform-independent framework for communication between servers and servlets.
- The servlet container implements and is compliant with the Servlet API specification.
- Developing servlets involves understanding how this communication takes place through the Servlet API.
- The Servlet API is a part of J2EE (Java 2 Enterprise Edition) and is comprised of two packages: `javax.servlet` and `javax.servlet.http`.
  - The `javax.servlet.http` package provides support for HTTP servlets.
  - Support for HTTP servlets is most prevalent.

Key Classes and Interfaces of the Servlet API

```
javax.servlet
  |   INTERFACE
  |   Servlet
  |   abstract
  |   GenericServlet
  | INTERFACE
  | ServletException
  | INTERFACE
  | ServletRequest
  | INTERFACE
  | ServletResponse
  | INTERFACE
  | ServletConfig
  | INTERFACE
  | ServletContext

javax.servlet.http
  | abstract
  | HttpServlet
  | INTERFACE
  | HttpServletRequest
  | INTERFACE
  | HttpServletResponse
  | INTERFACE
  | HttpSession
  | INTERFACE
  | Cookie
```

...
### The `javax.servlet` Package

<table>
<thead>
<tr>
<th>Class/Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interface Servlet</td>
<td>The main interface that every servlet must implement. It defines the key methods that a servlet container calls to control the servlet.</td>
</tr>
<tr>
<td>abstract class GenericServlet</td>
<td>This abstract class can be used as the starting point for implementing servlets. In particular, it implements all the methods of the Servlet interface except the service() method. This abstract class also implements the ServletConfig interface which allows the servlet container to pass information to the servlet.</td>
</tr>
<tr>
<td>interface ServletRequest</td>
<td>This interface provides the methods to extract information from a client request.</td>
</tr>
<tr>
<td>interface ServletResponse</td>
<td>This interface provides the methods to create and send an appropriate response to a client request.</td>
</tr>
<tr>
<td>interface ServletConfig</td>
<td>This interface which allows the servlet container to pass information to a servlet.</td>
</tr>
<tr>
<td>interface ServletContext</td>
<td>This interface allows the servlet to communicate with its container.</td>
</tr>
<tr>
<td>class ServletException</td>
<td>A general exception class to signal servlet runtime errors.</td>
</tr>
</tbody>
</table>

### The `javax.servlet.http` Package

<table>
<thead>
<tr>
<th>Class/Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract class HttpServlet</td>
<td>This abstract class extends the GenericServlet class and is used for implementing HTTP servlets, i.e. servlets which use HTTP for requests and responses. In particular, it provides stubs for the doHttpRequestMethodName() methods which correspond to the HTTP method used in the request (GET, POST, HEAD, etc.). A concrete servlet can override the appropriate methods to handle the different HTTP request methods.</td>
</tr>
<tr>
<td>interface HttpServletRequest</td>
<td>This interface extends the ServletRequest interface to handle HTTP requests.</td>
</tr>
<tr>
<td>interface HttpServletResponse</td>
<td>This interface extends the ServletResponse interface to create and send an appropriate HTTP response to an HTTP request.</td>
</tr>
<tr>
<td>interface HttpSession</td>
<td>This interface provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.</td>
</tr>
<tr>
<td>class Cookie</td>
<td>This class provides support for cookies to be used in requests and responses.</td>
</tr>
</tbody>
</table>
How the objects in the above diagram function and interact is essential to developing and deploying servlets.

Advantages of Servlets to extend Server Functionality

- **Flexibility**
  - A new servlet can be plugged in without changing the server.

- **Separation of Responsibilities**
  - The server takes care of the network connections and communication.
  - The servlets take care of application logic: generating appropriate responses to client requests.

- **Compiled language implementation**
  - Advantages of a compiled high-level language as compared to a scripting language like Perl.

- **Scalability**
  - Multi-threaded servlets more efficient than process-based scripts in handling large number of client requests.

- **Portability**
  - A servlet can be ported to another web server as long as the server provides a compatible servlet container.

- **More flexible and less Security Restrictions than Applets**
### Request Handling Cycle

1. Sends an HTTP request
2. Passes request to servlet container
3. Instantiates a thread to process each request.
4. Creates an HTTP response.
5. Returns HTTP response to web server.
6. Returns HTTP response to client browser.

### Servlet Life Cycle Methods

- The abstract `javax.servlet.http.HttpServlet` class provides the key methods *invoked on the servlet by the servlet container* during the servlet’s life time.

**void init(ServletConfig config)**

- The servlet container invokes this method only once on the servlet after the servlet class has been loaded and instantiated, supplying it a `ServletConfig` object.
- A servlet can override the `init()` method (without any parameters) to do its own initialization.

**void service(HttpServletRequest request, HttpServletResponse response)**

- The servlet container invokes the first method on the servlet for each request received for this servlet.
- Information about the request is contained in the `HttpServletRequest` object, and the servlet can deliver a suitable response using the `HttpServletResponse` object.
- The first method calls the second method which dispatches the request to an appropriate `doHttpRequestMethodName()` method depending on the HTTP method used in the request.
- A servlet should override the appropriate `doHttpRequestMethodName()` methods.

**void destroy()**

- The servlet container invokes this method on the servlet when the servlet is no longer needed.
- The servlet can no longer process requests after it is destroyed.
- A servlet can override this method to execute cleanup code.
HTTP Request Handling Methods

- The abstract `javax.servlet.http.HttpServlet` class provides default implementation for the `doHttpServletRequestMethodName()` methods which have the following method prototype:

```java
protected void doHttpServletRequestMethodName( HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException;
```

<table>
<thead>
<tr>
<th>HTTP Request Method Name</th>
<th>HttpServlet Method Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>doGet</td>
</tr>
<tr>
<td>HEAD</td>
<td>doHead</td>
</tr>
<tr>
<td>POST</td>
<td>doPost</td>
</tr>
</tbody>
</table>

- Depending on the HTTP method in the request, the corresponding servlet method is invoked.
- The default implementation of a `doHttpServletRequestMethodName()` method returns a `HttpServletResponse.SC_BAD_REQUEST` response (error code 400).
- The servlet should override the appropriate request handling methods depending on the business logic.

Servlet States

- Before a servlet can process requests, it must be loaded by the servlet container.
- The servlet container loads and instantiates the servlet class.
- The servlet is initialized once during its life time. As part of the initialization, the `init()` method of the servlet is also called.
- Once instantiated and loaded, the servlet is ready for servicing requests. This ultimately involves one of the servlet methods `doHttpServletRequestMethodName()` to handle the request. Each request is executed in a separate thread.
- When no longer needed, the servlet is destroyed by the servlet container. Before destroying the servlet, the container calls the `destroy()` method of the servlet.
- Several threads can share the servlet instance. Thus, fields cannot be assumed to be thread-safe.
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class ServletSkeleton extends HttpServlet {
    public void init() throws ServletException { /* implementation */ } // (1)

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException
    { /* implementation */ } // (2)

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException
    { /* implementation */ } // (3)

    public void destroy() { /* implementation */ } // (4)

    public String getServletInfo() { /* implementation */ } // (5)
}

HTTP Request Methods Revisited

- The browser sends an HTTP request to a web server when such events as the following occur:
  - A hyperlink is clicked on a HTML page.
  - A form is filled on a HTML page and submitted.
  - A URL is specified for the browser to retrieve a resource.
- The browser uses the GET request method by default to send the request.
- A GET request has no entity-body, and therefore limited in the amount data it can send in the request.
  - A GET request is suitable for retrieving a passive resource.
  - A query string in a GET request specifies the parameters that are submitted in the request.
  - A GET request is therefore prone to a security risk.
- A POST request is suitable for retrieving an active resource, and can specify any additional information in the entity-body.
  - The entity-body in a POST request specifies the parameters that are submitted in the request.
Handling Servlet Requests: the HttpServletRequest Interface

- A (HttpServletRequest) object is created by the servlet container and passed to the servlet in the `service()` method, and subsequently to the appropriate `doHttpRequestMethodName()`.

The following methods supply protocol-, method- and path-related information in the request:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>String getMethod()</code></td>
<td>Returns the name of the HTTP method in this request. For example: GET, POST, or HEAD.</td>
</tr>
<tr>
<td><code>String getScheme()</code></td>
<td>Returns the name of the scheme used to make this request, for example, http or ftp.</td>
</tr>
<tr>
<td><code>String getProtocol()</code></td>
<td>Returns the name and version of the protocol used in the request. The form is <code>protocol/majorVersion.minorVersion</code>, for example, HTTP/1.1.</td>
</tr>
<tr>
<td><code>StringBuffer getRequestURL()</code></td>
<td>Reconstructs the URL the client used to make the request.</td>
</tr>
<tr>
<td><code>String getRequestURI()</code></td>
<td>Returns the part of URL in this request that follows the host and port number up to the query string in the method line.</td>
</tr>
<tr>
<td><code>String getServletPath()</code></td>
<td>Returns the part of the URL that calls the servlet.</td>
</tr>
<tr>
<td><code>String getServletContextPath()</code></td>
<td>Returns the portion of the request URI that indicates the context of the request. The context path always comes first in a request URI. The path starts with a &quot;/&quot; character but does not end with a &quot;/&quot; character.</td>
</tr>
</tbody>
</table>

```
GET http://localhost:8080/horoscope/SimpleHoroscope?sign=aquarius HTTP/1.0
```

Requested URL
### ATIJ Core Servlets

**String getQueryString()**
Returns the query string that is specified after the path.

**String getPathTranslated()**
Returns any extra path information after the servlet name but before the query string, and translating it to a real path.

**String getPathInfo()**
Returns any extra path information associated with the URL in this request.

---

- The following methods supply information about the **headers** in the request:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String getHeader(String name)</td>
<td>Returns the String value of the specified header name.</td>
</tr>
<tr>
<td>int getIntHeader(String name)</td>
<td>Returns the int value of the specified header name.</td>
</tr>
<tr>
<td>Enumeration getHeaders(String name)</td>
<td>Returns an enumerator of Strings containing all the values associated with the specified header name.</td>
</tr>
<tr>
<td>Enumeration getHeaderNames()</td>
<td>Returns an enumerator of Strings containing all the header names.</td>
</tr>
</tbody>
</table>

- The following methods supply information about the **client request parameters** submitted in the request:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String getParameter(String name)</td>
<td>Returns one value for the parameter name if specified in the request, null otherwise. Note a parameter name can have several values associated with it.</td>
</tr>
<tr>
<td>String[] getParameterValues(String name)</td>
<td>Returns all the values for the parameter name if specified in the request, null otherwise.</td>
</tr>
<tr>
<td>Enumeration getParameterNames()</td>
<td>Returns an enumerator of Strings containing the parameter names.</td>
</tr>
</tbody>
</table>
Other misc. methods for various kinds of information pertaining to the request:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String getServerName()</td>
<td>Returns the hostname of the server that received the request.</td>
</tr>
<tr>
<td>Cookies[] getCookies()</td>
<td>Returns an array with all the cookies sent in the request.</td>
</tr>
<tr>
<td>HttpSession getSession(boolean instantiate)</td>
<td>Returns the current session associated with the request if there is one. Otherwise creates one if argument instantiate is true.</td>
</tr>
<tr>
<td>HttpSession getSession()</td>
<td>This method is equivalent to getSession(true).</td>
</tr>
</tbody>
</table>

Creating Servlet Responses: the HttpServletResponse Interface

- A (HttpServletResponse object is created by the servlet container and passed to the servlet in the service() method, and subsequently to the appropriate doHttpRequestMethodName().
- The servlet uses the HttpServletResponse object to formulate a response to the request.
- The following methods can be used to set specific headers in the response:
  - void setStatus(int code) Sets the status code of the response.
  - void setContentType(String type) Sets the MIME content type to the specified type.
  - void setContentLength(int size) Sets the content length to the specified size.
  - void addCookie(Cookie cookie) Adds a cookie to the response.
- The HttpServletResponse interface provides constants for the HTTP status codes:
  - HttpServletResponse.SC_OK // HTTP Status-Code 200: OK
  - HttpServletResponse.SC_NOT_FOUND // HTTP Status-Code 404: Not Found
  - HttpServletResponse.SC_NOT_IMPLEMENTED // HTTP Status-Code 501: Not Implemented
The following methods can be used to set any *headers* in the response:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void setHeader(String name, String value)</code></td>
<td>Sets the specified response header name to the specified value.</td>
</tr>
<tr>
<td><code>void addHeader(String name, String value)</code></td>
<td>Adds the specified value to the specified multi-valued response header name.</td>
</tr>
<tr>
<td><code>void setDateHeader(String name, long msecs)</code></td>
<td>Sets the specified response header name to the date value specified in milliseconds.</td>
</tr>
<tr>
<td><code>void addDateHeader(String name, long msecs)</code></td>
<td>Adds the date value in milliseconds to the specified multi-valued response header name.</td>
</tr>
<tr>
<td><code>void setIntHeader(String name, int value)</code></td>
<td>Sets the specified response header name to the int value.</td>
</tr>
<tr>
<td><code>void addIntHeader(String name, int value)</code></td>
<td>Adds the int value to the specified multi-valued response header name.</td>
</tr>
</tbody>
</table>

Other Misc. Methods for Handling the Response:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>PrintWriter getWriter()</code></td>
<td>Returns a <code>PrintWriter</code> that can send characters in the entity-body of the response. Calling the <code>flush()</code> method on the writer commits the response.</td>
</tr>
<tr>
<td><code>ServletOutputStream getOutputStream()</code></td>
<td>Returns a <code>ServletOutputStream</code> that can send binary data in the entity-body of the response. Calling the <code>flush()</code> method on the writer commits the response. Note that both a writer and a servlet output stream cannot be used for the entity-body of the response, only one of them.</td>
</tr>
<tr>
<td><code>void sendRedirect(String location)</code></td>
<td>Sends a temporary redirect response to the client using the specified redirect location URL.</td>
</tr>
<tr>
<td><code>void sendError(int statusCode)</code></td>
<td>Send the specified error code in the response, and the message if it is specified.</td>
</tr>
<tr>
<td><code>void sendError(int statusCode, String message)</code></td>
<td></td>
</tr>
</tbody>
</table>
Deployment Structure for Web Applications

- All web applications are placed in this directory

Web Application Structure:
- Publicly accessible files
- All information and support files
- Class files in their package structure
- Other necessary JAR archives
- Deployment descriptor file

Implementing a Servlet

- When overriding a `doHttpRequestMethodName()` method, the following procedure is recommended:
  - Read the request data.
  - Write the response headers.
  - Get the response’s writer or output stream object.
  - Write the response data.
- In the response, include content type and encoding.
- When using a `PrintWriter` object to return the response, set the content type before accessing the `PrintWriter` object.
- The servlet container must write the headers before committing the response, because in HTTP the headers must be sent before the response body.
  - Committing locks some of the features of the servlet, this is specially true with respect to the response.
- A common strategy is to write the response data to an internal stream, and dump this data out to the output stream which is only closed after all data has been flushed.
- Since several threads (one for each request) can be executing the servlet, normal thread-safety precautions apply.
Developing, Deploying and Running a Web Application

1. Implement the classes of the web application (StopTheWorld.java):
   ```java
   import java.io.*;
   import javax.servlet.*;
   import javax.servlet.http.*;

   public class StopTheWorld extends HttpServlet {
     public void doGet(HttpServletRequest request, HttpServletResponse response)
       throws ServletException, IOException {
       String msg = "Stop the world! I want to get off!";
       // (1) Set the content type of the response
       response.setContentType("text/html");
       // (2) Get a writer for the response.
       PrintWriter responseWriter = response.getWriter();
       // (3) Compose the HTML content (entity-body) for the response.
       responseWriter.println("<html><body>" + msg + "</h3>";
       responseWriter.println("</body></html>");
       responseWriter.flush(); // Commit the response.
       responseWriter.close();
     }
   }
   ```

2. Specify the deployment descriptor for the web application (web.xml).
   ```xml
   <?xml version="1.0" encoding="ISO-8859-1"?>
   <web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
     http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
     <display-name>myExamples</display-name>
     <!-- A servlet element -->
     <servlet>
       <servlet-name>mySimpleServlet</servlet-name>
       <servlet-class>StopTheWorld</servlet-class>
     </servlet>
     <!-- A servlet mapping element -->
     <servlet-mapping>
       <servlet-name>mySimpleServlet</servlet-name>
       <url-pattern>/SimpleServlet/*</url-pattern>
     </servlet-mapping>
   </web-app>
   ```

   A servlet element is specified for each servlet in the web application. The servlet name is used to identify pertinent information about the servlet, for example its class.

   A servlet mapping element is specified for each servlet in the web application. The url pattern /SimpleServlet/* is mapped to the servlet named mySimpleServlet. The url pattern need not be the same as the name of the servlet.

   The order in which the elements are specified is important.
3. Install the web application in the webapps directory.
   – The web application is installed in a new directory called myExamples.
   – The myExamples directory is the root document of this web application.
   – The web application structure describes the application to the servlet container.

   `<Web server home>`
   ```
   ... webapps
   ... Web Application Structure
   | myExamples  Document Root
   | WEB-INF     All information and support files
   | classes    Class files in their package structure
   | StopTheWorld.class
   | web.xml    Deployment descriptor file
   ```

4. Configuring the web application.
   – The context of a web application must be specified for each web application.

   – For example, the myExamples web application has the following context:

   `<Context
   path="/myExamples"
docBase="/myExamples"
reloadable="true"
workDir="/webapps/myExamples/work">
</Context>

Two option for specifying the context information:
1. The server.xml file in the the TOMCAT_HOME\conf directory can be modified to provide the context information for each web application.
2. A context file can be created in the TOMCAT_HOME\conf\Catalina\localhost directory for each web application (preferred solution).
   - For example, the file myExamples.xml with the context information above would be placed in the TOMCAT_HOME\conf\Catalina\localhost directory.
5. Run the web application in a browser.
   – The web server must be up and running.
   – Enter an appropriate URL in the browser to send a request to the SimpleServlet.

   ![Image of a web browser with a request to SimpleServlet]

   • If the class files or the deployment descriptor file change, the web application structure must be updated with the new files.
     – If the application is reloadable, Tomcat will update the files.
     – However, it might be necessary to stop and restart the web application, and at times to restart the web server.

---

**HTML Forms: Sending Data in a HTTP Request**

- The HTML FORM element is an entity for submitting data in a HTTP request (See SimpleHoroscope.html in the horoscope web-app.).

```html
<form name="signs" action="SimpleHoroscope" method="post">
  <input type="radio" name="sign" value="aquarius" checked="checked"/>
  Aquarius (Jan 21 - Feb 19) <br/>
  <input type="radio" name="sign" value="pisces"/>
  Pisces (Feb 20 - Mar 20) <br/>
  ...
  <input type="submit" value="Read the stars!"/>
</form>
```

- The attributes in the FORM element:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the form.</td>
</tr>
<tr>
<td>action</td>
<td>The resource to process the submitted data. The servlet path is mapped to the servlet which will receive the data from the form in a HTTP request.</td>
</tr>
<tr>
<td>method</td>
<td>The HTTP method to use to submit the request. Default method is the GET method, so the POST method must explicitly be specified.</td>
</tr>
</tbody>
</table>
• Each INPUT element specifies properties of the input control using these attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Specifies the type of control. The value &quot;radio&quot; indicates a radio button, meaning that only one such control can be checked in the form. The checked tag means that this button will initially appear checked in the page. Any text appearing after the INPUT element is used as label for the radio button. The value &quot;submit&quot; indicates a submit button. Some other values are &quot;text&quot; (text field) and &quot;password&quot; (password field).</td>
</tr>
<tr>
<td>name</td>
<td>This attribute specifies the name of the input control. Each radio button has the same name (&quot;sign&quot;).</td>
</tr>
<tr>
<td>value</td>
<td>For a radio button, the value of this attribute is associate with the value in the name attribute, forming a parameter-value pair. For a submit button, specifies the label of the button.</td>
</tr>
</tbody>
</table>

– For other examples of INPUT element, see HoroscopeWithSelect.html and HoroscopeWithTextField.html.

• Clicking the submit control results in the values of the name and value attributes of the selected radio button to be submitted in the request.

  – For example, if the aquarious radio button was selected, clicking the submit button would be analogous to submitting the query string "sign=aquarious".

  – Note that all radio buttons have the same name ("sign") but the value attributes have different values depending on the star sign.

HTML Rendering of a Form

Note the HTML content of the file is returned by the web-server, not the servlet.

Get Your Daily Horoscope (One-size Fits All)

Learn about all the calamities that can come your way.

Of course, it's all FREE! Including the calamities.

Choose your sign:

- Aquarius (Jan 21 - Feb 19)
- Pisces (Feb 20 - Mar 20)
- Aries (Mar 21 - Apr 20)
- Taurus (Apr 21 - May 20)
- Gemini (May 21 - Jun 20)
- Cancer (Jun 21 - Jul 22)
- Leo (Jul 23 - Aug 22)
- Virgo (Aug 23 - Sep 22)
- Libra (Sep 23 - Oct 22)
- Scorpio (Oct 23 - Nov 21)
- Sagittarius (Nov 22 - Dec 21)
- Capricorn (Dec 22 - Jan 20)

Read the stars!
Example: Implementing Simple Horoscope

```java
public class SimpleHoroServlet extends HttpServlet {

    protected String servletName = "Simple Horoscope Servlet";

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

        // Get the sign value
        String sign = req.getParameter("sign").toLowerCase();
        String horoscopeReply = getHoroscope(sign);
        if (horoscopeReply == null)
            horoscopeReply = "Looks like you were born under " + "an as-yet undiscovered star."
;
        // Set any response headers
        resp.setContentType("text/html");

        // Get a writer for the response
        PrintWriter out = resp.getWriter();

        out.println("<html><body><h1>" + servletName + "</h1>";
        out.println("<h3> Your horoscope, " + sign.toUpperCase() + ":</h3>";
        out.println("<h4>" + horoscopeReply + "</h4>");
        out.println("<h4>Good luck! You are going to need it.");
        out.println("</body></html>");

        // Flush and close the writer.
        out.flush();
        out.close();
    }

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException { doPost(req, resp); }

    protected String getHoroscope(String sign) {
        return "If you think knowing your horoscope " + "is going to help you, you are gravely mistaken.";
    }
}
```
Information in Deployment Descriptor (file web.xml)

...<servlet>
   <servlet-name>SimpleHoroscope</servlet-name>
   <servlet-class>SimpleHoroServlet</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>SimpleHoroscope</servlet-name>
   <url-pattern>/SimpleHoroscope</url-pattern>
</servlet-mapping>
...

GET Method in the HTTP Request
Reading Request Method and Paths Information

- See file ServletUtil.java.

```java
public static void echoRequestMethodLineInfo(HttpServletRequest req,
                                              PrintWriter out)
    throws ServletException, IOException {
    out.print("<h3>Request Method- and Path-related Information</h3>");
    out.print("<p>");
    out.print("Method: " + req.getMethod() + "<br>");
    out.print("Scheme: " + req.getScheme() + "<br>");
    out.print("Protocol: " + req.getProtocol() + "<br>");
    out.print("Requested URL: " + req.getRequestURL() + "<br>");
    out.print("Requested URI: " + req.getRequestURI() + "<br>");
    out.print("Servlet Path: " + req.getServletPath() + "<br>");
    out.print("Context Path: " + req.getContextPath() + "<br>");
    out.print("Query: " + req.getQueryString() + "<br>");
    out.print("Path info: " + req.getPathInfo() + "<br>");
    out.print("Path translated: " + req.getPathTranslated() + "<br>");
    out.println("</p>");
}
```

Reading Request Headers

- See file ServletUtil.java.

```java
public static void echoRequestHeaders(HttpServletRequest req,
                                        PrintWriter out)
    throws ServletException, IOException {
    out.print("<h3>Request Header Information</h3>");
    out.print("<p>");
    Enumeration enumerator = req.getHeaderNames();
    while (enumerator.hasMoreElements()) {
        String header = (String) enumerator.nextElement();
        String value = req.getHeader(header);
        out.print(header + ": " + value + "<br>");
    }
    out.println("</p>");
}
```
Reading Request Parameters

- See file ServletUtil.java.

```java
public static void echoRequestParameters(HttpServletRequest req, PrintWriter out) throws ServletException, IOException {
    out.println("<h3>Request Parameters</h3>");
    Enumeration enumerator = req.getParameterNames();
    while (enumerator.hasMoreElements()) {
        String paramName = (String) enumerator.nextElement();
        String paramValue = req.getParameter(paramName);
        out.print(paramName + " : " + paramValue + " <br> ");
    }
    out.println("</p>");
}
```

HTTP Request

Illustrating:
- Method- and Path-related Information
- Request Headers
- Request Parameters

Request Method- and Path-related Information

- Method: POST
- Scheme: http
- Protocol: HTTP/1.1
- Requested URL: http://localhost:8080/horoscope/SimpleHoroscope
- Requested URI: /horoscope/SimpleHoroscope
- Servlet Path: SimpleHoroscope
- Context Path: /horoscope
- Query: null
- Path: null
- Path Translated: null

Request Header Information

- `accept`: */*
- `referrer`: http://localhost:8080/horoscope/SimpleHoroscope.html
- `content-type`: application/x-www-form-urlencoded
- `accept-encoding`: gzip, deflate
- `user-agent`: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
- `host`: localhost:8080
- `content-length`: 19
- `connection`: Keep-Alive
- `cache-control`: no-cache
- `cookie`: JSESSIONID=4F4D7EC12F08CF84DA00E21BAE39CC1

Request Parameters

- sign: aquarius
Files for Simple Horoscope Web Application

The *index.html* file is a convenient way to group a set of servlets deployed under the same web application.

The * *.html* files are placed directly under the document root.

Useful Methods of the Abstract Servlet/HttpServlet Class

- **ServletConfig getServletConfig()**: A ServletConfig object provides access to the servlet’s configuration information: initialization parameters and servlet context. (See also the API for the ServletConfig interface).
  - Such an object is passed to the servlet at initialization.
  - A servlet can invoke this method to obtain the servlet configuration associated with the servlet.
  - This method is specified in the Servlet interface.

- **ServletContext getServletContext()**: A ServletContext object provides communication between the servlet and the servlet container. (See also the API for the ServletContext interface).
  - A servlet can invoke this method to obtain the servlet context associated with the servlet.
  - This method is specified in the ServletConfig interface.

- **String getServletInfo()**: A servlet implements this method to provide such information as author, version, and copyright information.
  - Servlet container can obtain this information by invoking this method on the servlet.
  - This method is specified in the Servlet interface.

- Note that a servlet also implements the ServletConfig interface.
Servlet Configuration: the ServletConfig Interface

- A ServletConfig object is used by the servlet container to pass information to the servlet during initialization.
- The servlet can call the following methods of the ServletConfig object to retrieve information about parameters specified in the deployment descriptor and to obtain the servlet context for the servlet.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String getInitParameter(String name)</td>
<td>Returns the value of the parameter name, or null if no such parameter is specified in its deployment descriptor.</td>
</tr>
<tr>
<td>Enumeration getInitParameterNames()</td>
<td>Returns a String enumerator over the names of all the parameters specified in the deployment descriptor.</td>
</tr>
<tr>
<td>String getServletName()</td>
<td>Returns the name of the servlet as specified in its deployment descriptor.</td>
</tr>
<tr>
<td>ServletContext getServletContext()</td>
<td>Returns the servlet context of the servlet.</td>
</tr>
</tbody>
</table>

Servlet Context: the ServletContext Interface

- A ServletContext object can be used by the servlet to interact with its environment.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputStream getResourceAsStream(String path)</td>
<td>Get an input stream connected to the passive resource specified by the path.</td>
</tr>
<tr>
<td>URL getResource(String path)</td>
<td>Get an URL for the passive resource which is mapped to the path. The path is relative to the document root of the web application.</td>
</tr>
<tr>
<td>String getRealPath(String virtualPath)</td>
<td>Returns a String containing the real path for the specified virtual path. For example, the path /help.html returns the platform-dependent absolute file path on the server's file system that would be served by a request for <a href="http://host/contextPath/help.html">http://host/contextPath/help.html</a>, where contextPath is the context path of this ServletContext.</td>
</tr>
<tr>
<td>void log(String msg)</td>
<td>Logs the specified message to a special servlet file (log file).</td>
</tr>
</tbody>
</table>

- The ServletContext object is contained within the ServletConfig object, which the server provides the servlet when the servlet is initialized.
- For a servlet container that is in a single JVM, every web application running in the container has exactly one servlet context.
  This servlet context is shared by the web components of the web application.
Simple Horoscope II: Overriding the `init()` Method

- The `init()` method can be overridden to setup whatever resources (database connections, files, etc.) the servlet needs for its operation.
- The Simple Horoscope is extended to use horoscope data from a specific file.
- The `init()` method ensures that the horoscope data has been read in from the file before handling client requests.
- We use a separate class `HoroscopeDataFile` to read the horoscope data files and create a map of `<sign, horoscope>` entries.
- The real path of the file is obtained through the servlet context.
  - Access to resources under the document root should be managed through the servlet context.

```java
public class HoroServletWithHardWiredFile extends SimpleHoroServlet {
    private HoroscopeDataFile horoscopeData;
    public void init() {
        try {
            servletName = "Simple Horoscope II (With Hard Wired Filename)";

            // File with horoscope data.
            String filename = "data/horoscope.data"; // Filename is hardwired.
            System.out.println("Filename: " + filename);

            // Get the real path of the file.
            ServletContext context = getServletContext();
            String realPath = context.getRealPath(filename);
            System.out.println("Real path: " + realPath);
            horoscopeData = new HoroscopeDataFile(realPath);
        } catch(IOException ioe) {
            System.err.println("I/O Error getting horoscope data");
        }
    }

    protected String getHoroscope(String sign) {
        return horoscopeData.getHoroscope(sign);
    }
}
```
The Horoscope File

// Horoscope data
public class HoroscopeDataFile {

    // The signs.

    // Map for <star sign, horoscope> entries.
    private TreeMap<String, String> horoscopeData;

    private String horoscopeFilename;

    public HoroscopeDataFile(String filename) throws IOException {
        horoscopeFilename = filename;
        horoscopeData = new TreeMap<String, String>();
        initHoroscope();
    }

    private void initHoroscope() throws IOException {
        BufferedReader source =
            new BufferedReader(
                new InputStreamReader(  
                    new FileInputStream(horoscopeFilename)));
        for (int i = 0; i < horoscopeSign.length; i++) {
            String txtLine = source.readLine();
            if (txtLine == null)
                break;
            horoscopeData.put(horoscopeSign[i], txtLine);
        }
        source.close();
    }

    public String getHoroscope(String sign) {
        return (String) horoscopeData.get(sign);
    }

    // ...
}
Running Simple Horoscope II

Get Your Daily Horoscope (using data file)
Learn about all the calamities that can come your way.
Of course, it’s all FREE! Including the calamities.

Choose your sign:
- Aquarius (Jan 21 - Feb 19)
- Pisces (Feb 20 - Mar 20)
- Aries (Mar 21 - Apr 20)
- Taurs (Apr 21 - May 21)
- Gemini (May 22 - Jun 21)
- Cancer (Jun 22 - Jul 22)
- Leo (Jul 23 - Aug 22)
- Virgo (Aug 23 - Sept 22)
- Libra (Sept 23 - Oct 23)
- Scorpio (Oct 24 - Nov 21)
- Sagittarius (Nov 22 - Dec 21)
- Capricorn (Dec 22 - Jan 19)

Simple Horoscope II (With Hard Wired Filename)

Your horoscope, AQUARIUS:
Expect someone to cause a bit of a rift that could result in a major change of plans. Don’t listen to this person’s plea for help. You will only get more grief.

Good luck! You are going to need it.

Simple Horoscope III: Specifying Initialization Parameters

- Instead of hardwiring the horoscope data file in the servlet, the filename can be specified in the deployment descriptor as an initialization parameter.
- Defining initialization parameters for Simple Horoscope III in the deployment descriptor web.xml:

```xml
...<servlet>
  <servlet-name>HoroscopeInitParams</servlet-name>
  <servlet-class>HoroServletInitParams</servlet-class>

  <!-- Define initialization parameters -->
  <init-param>
    <param-name>HoroscopeDatafile</param-name>
    <param-value>data/horoscope.data</param-value>
  </init-param>
</servlet>
...
```

- The init-param element specifies parameter HoroscopeDatafile having the value data/horoscope.data for the servlet named HoroscopeInitParams.
- Note that init-param element is a part of the servlet element.
Simple Horoscope III: Reading Initialization Parameters

- Initialization parameters can be read conveniently in the `init()` method as part of the servlet's loading process.

```java
public class HoroServletInitParams extends SimpleHoroServlet {
  protected HoroscopeDataFile horoscopeData;
  public void init() {
    try {
      servletName = "Simple Horoscope III (With Parameterized Filename)";
      // Read the parameter from the deployment descriptor.
      String filename = getInitParameter("HoroscopeDatafile");
      System.out.println("Filename: " + filename);

      // Get the real path of the file.
      ServletContext context = getServletContext();
      String realPath = context.getRealPath(filename);
      System.out.println("Real path: " + realPath);
      horoscopeData = new HoroscopeDataFile(realPath);
    } catch (IOException ioe) {
      System.err.println("I/O Error getting horoscope data");
    }
  }

  protected String getHoroscope(String sign) {
    return horoscopeData.getHoroscope(sign);
  }
}
```

Servlet Configuration: Reading Initialization Parameters

- The servlet configuration provides access to the initialization parameters.
- See file ServletUtil.java.

```java
public static void echoInitParameters(ServletConfig config, PrintWriter out)
  throws ServletException, IOException {
  out.print("<h3>Initialization Parameters</h3>");
  out.print("<p>");
  Enumeration enumerator = config.getInitParameterNames();
  while (enumerator.hasMoreElements()) {
    String paramName = (String) enumerator.nextElement();
    String paramValue = config.getInitParameter(paramName);
    out.print(paramName + ": " + paramValue + "<br>");
  }
  out.println("</p>");
}
```
Running Simple Horoscope III

Get Your Daily Horoscope (illustrating initialization parameters)

Learn about all the climates.

Of course, it's all FREE! In

Choose your sign:

- Aquarius (Jan 21 - Feb 19)
- Pisces (Feb 20 - Mar 20)
- Aries (Mar 21 - Apr 20)
- Taurus (Apr 21 - May 21)
- Gemini (May 22 - Jun 21)
- Cancer (Jun 22 - Jul 23)
- Leo (Jul 24 - Aug 23)
- Virgo (Aug 24 - Sept 23)
- Libra (Sept 24 - Oct 23)
- Scorpio (Oct 24 - Nov 22)
- Sagittarius (Nov 23 - Dec 21)
- Capricorn (Dec 22 - Jan 20)

Simple Horoscope III (With Parameterized Filename)

Request Parameters

sign: virgo

Initialization Parameters

HoroscopeDateFile: data/horoscope.data

Your horoscope, VIRGO:

Someone is talking behind your back. You will have difficulty avoiding arguments and controlling your temper. Don't expect positive changes at home.

Good luck! You are going to need it.