detalladas para crear el programa. [*Nota:* el marcado en la figura 26.4 y en los demás listados de archivos JSP en este capítulo es el mismo que el marcado que aparece en Java Studio Creator 2, pero hemos cambiado el formato de estos listados para fines de presentación, para que el código sea más legible].

Java Studio Creator 2 genera todo el marcado que se muestra en la figura 26.4 cuando establecemos el título de la página Web, arrastramos dos componentes **Texto estático** en la página y establecemos las propiedades de estos componentes. Los componentes `Texto estático` muestran texto que el usuario no puede editar. En breve le mostraremos estos pasos.

```
1  <?xml version = "1.0" encoding = "UTF-8"?>
2
3  <!-- Fig. 26.4: Hora.jsp -->
4  <!-- Archivo JSP generado por Java Studio Creator 2, que muestra -->
5  <!-- la hora actual en el servidor Web -->
6  <jsp:root version = "1.2"
7     xmlns:f = "http://java.sun.com/jsf/core"
8     xmlns:h = "http://java.sun.com/jsf/html"
9     xmlns:jsp = "http://java.sun.com/JSP/Page"
10    xmlns:ui = "http://www.sun.com/web/ui">
11    <jsp:directive.page contentType = "text/html;charset = UTF-8"
12       pageEncoding = "UTF-8"/>
13    <f:view>
14       <ui:page binding = "#{Hora.page1}" id = "page1">
15          <ui:html binding = "#{Hora.html1}" id = "html1">
16             <ui:head binding = "#{Hora.head1}" id = "head1"
17                title = "Hora Web: un ejemplo simple">
18                <ui:link binding = "#{Hora.link1}" id = "link1"
19                   url = "/resources/stylesheet.css"/>
20                </ui:head>
21                <ui:meta content = "60" httpEquiv = "refresh" />
22                <ui:body binding = "#{Hora.body1}" id = "body1"
23                   style = "-rave-layout: grid">
24                   <ui:form binding = "#{Hora.form1}" id = "form1">
25                      <ui:staticText binding = "#{Hora.encabezadoHora}" id =
26                         "encabezadoHora" style = "font-size: 18px; left: 24px;
27                         top: 24px; position: absolute" text = "Hora actual
28                         en el servidor Web : "/>
29                      <ui:staticText binding = "#{Hora.textoReloj}" id =
30                         "textoReloj" style = "background-color: black;
31                         color: yellow; font-size: 18px; left: 24px; top:
32                         48px; position: absolute"/>
33                   </ui:form>
34             </ui:body>
35          </ui:html>
36       </ui:page>
37    </f:view>
38 </jsp:root>
```

Figura 26.4 | Archivo JSP generado por Java Studio Creator 2, que muestra la hora actual en el servidor Web.

26.5.1 Análisis de un archivo JSP

Los archivos JSP que se utilizan en este ejemplo (y los siguientes) se generan casi completamente mediante Java Studio Creator 2, el cual proporciona un Editor visual que nos permite crear la GUI de una página al arrastrar y soltar componentes en un área de diseño. El IDE genera un archivo JSP en respuesta a las interacciones del programador. En la línea 1 de la figura 26.4 está la declaración XML, la cual indica que la JSP está expresada en sintaxis XML, junto con la versión de XML que se utiliza. En las líneas 3 a 5 hay comentarios que agregamos a la JSP, para indicar su número de figura, nombre de archivo y propósito.

En la línea 6 empieza el elemento raíz para la JSP. Todas las JSPs deben tener este elemento `jsp:root`, el cual tiene un atributo `version` para indicar la versión de JSP que se está utilizando (línea 6), y uno o más atributos `xmlns` (líneas 7 a 10). Cada **atributo `xmlns`** especifica un prefijo y un URL para una biblioteca de etiquetas, lo cual permite a la página usar las etiquetas especificadas en esa biblioteca. Por ejemplo, la línea 9 permite a la página usar los elementos estándar de las JSPs. Para usar estos elementos, hay que colocar el prefijo `jsp` antes de la etiqueta de cada elemento. Todas las JSPs generadas por Java Studio Creator 2 incluyen las bibliotecas de etiquetas especificadas en las líneas 7 a 10 (la biblioteca de componentes JSF básicos, la biblioteca de componentes JSF de HTML, la biblioteca de componentes JSP estándar y la biblioteca de componentes JSF de interfaz de usuario).

En las líneas 11 y 12 se encuentra el elemento `jsp:directive.page`. Su atributo `contentType` especifica el tipo MIME (`text/html`) y el conjunto de caracteres (`UTF-8`) que utiliza la página. El atributo `pageEncoding` especifica la codificación de caracteres que utiliza el origen de la página. Estos atributos ayudan al cliente (por lo general, un navegador Web) a determinar cómo desplegar el contenido.

Todas las páginas que contienen componentes JSF se representan en un **árbol de componentes** (figura 26.5) con el elemento JSF raíz `f:view`, que es de tipo `UIViewRoot`. Para representar la estructura de este árbol de componentes en una JSP, se encierran todas las etiquetas de los componentes JSF dentro del elemento `f:view` (líneas 13 a 37).

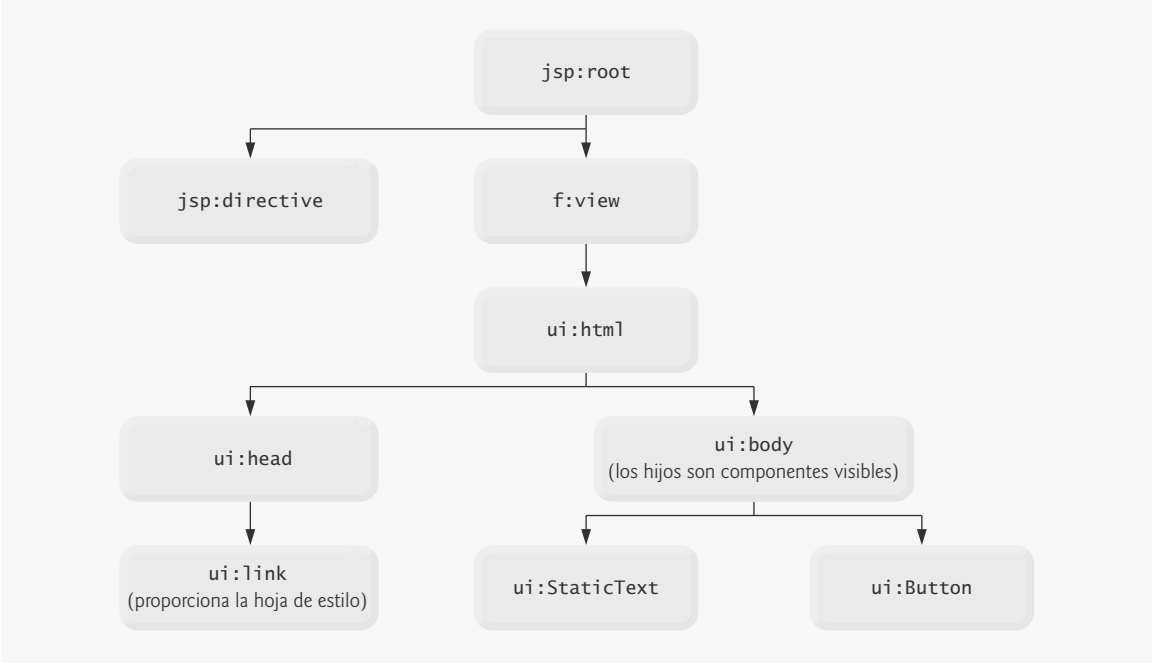

Figura 26.5 | Árbol de componentes JSF de ejemplo.

En las líneas 14 a 20 empieza la definición de la JSP con las etiquetas `ui:page`, `ui:html` y `ui:head`, todas de la biblioteca de etiquetas `ui` (componentes JSF de interfaz de usuario). Éstos y muchos otros elementos de página tienen un atributo `binding`. Por ejemplo, el elemento `ui:head` (línea 16) tiene el atributo `binding = "#{Hora.head}."`. Este atributo utiliza la notación del **Lenguaje de expresiones JSF** (es decir, `#{Hora.head}`) para hacer referencia a la propiedad `head` en la clase `Hora` que representa al bean de página (en la figura 26.6 podrá ver esta clase). Es posible enlazar un solo atributo de un elemento JSP a una propiedad en cualquiera de los JavaBeans de la aplicación Web. Por ejemplo, el atributo `text` de un componente `ui:label` se puede enlazar a una propiedad `String` en el objeto `SessionBean` de la aplicación. En la sección 26.7.2 veremos un ejemplo de esto.

El elemento `ui:head` (líneas 16 a 20) tiene un atributo `title` que especifica el título de la página. Este elemento también contiene un elemento `ui:link` (líneas 18 y 19), el cual especifica la hoja de estilo CSS que utiliza la página. El elemento `ui:body` (líneas 22 a 34) contiene un elemento `ui:form` (líneas 24 a 33), el cual contiene

dos componentes `ui:staticText` (líneas 25 a 28 y 29 a 32). Estos componentes muestran el texto de la página. El componente `encabezadoHora` (líneas 25 a 28) tiene un atributo `text` (líneas 27 y 28) que especifica el texto a mostrar (es decir, `"Hora actual en el servidor Web:"`). El componente `textoReloj` (líneas 29 a 32) no especifica un atributo de `texto`, ya que el texto de este componente se establecerá mediante programación.

Para que el marcado en este archivo se muestre en un navegador Web, todos los elementos de la JSP se asignan automáticamente a elementos de XHTML que el navegador reconoce. El mismo componente Web se puede asignar a varios elementos de XHTML distintos, dependiendo del navegador Web cliente y de las configuraciones de las propiedades del componente. En este ejemplo, los componentes `ui:staticText` (líneas 25 a 28, 29 a 32) se asignan a elementos `span` de XHTML. Un elemento `span` contiene texto que se muestra en una página Web, y que comúnmente se utiliza para controlar el formato del texto. Los atributos `style` de un elemento `ui:staticText` de una JSP se representan como parte del correspondiente atributo `style` del elemento `span` cuando el navegador despliega la página. En un momento le mostraremos el documento XHTML que se produce cuando un navegador solicita la página `Hora.jsp`.

26.5.2 Análisis de un archivo de bean de página

En la figura 26.6 se presenta el archivo de bean de página. En la línea 3 se indica que esta clase pertenece al paquete `horaweb`. Esta línea se genera automáticamente y especifica el nombre del proyecto como el nombre del paquete. En la línea 17 empieza la declaración de la clase `Hora` e indica que hereda de la clase `AbstractPageBean` (del paquete `com.sun.rave.web.ui.appbase`). Todas las clases de bean de página que soportan archivos JSP con componentes JSF deben heredar de la clase abstracta `AbstractPageBean`, la cual proporciona métodos para el ciclo de vida de las páginas. Observe que el IDE hace que el nombre de la clase coincida con el nombre de la página. El paquete `com.sun.rave.web.ui.component` incluye clases para muchos de los componentes JSF básicos (vea las instrucciones `import` en las líneas 6 a 11 y 13).

```
1  // Fig. 26.6: Hora.java
2  // Archivo de bean de página que establece textoReloj a la hora en el servidor Web.
3  package horaweb;
4
5  import com.sun.rave.web.ui.appbase.AbstractPageBean;
6  import com.sun.rave.web.ui.component.Body;
7  import com.sun.rave.web.ui.component.Form;
8  import com.sun.rave.web.ui.component.Head;
9  import com.sun.rave.web.ui.component.Html;
10 import com.sun.rave.web.ui.component.Link;
11 import com.sun.rave.web.ui.component.Page;
12 import javax.faces.FacesException;
13 import com.sun.rave.web.ui.component.StaticText;
14 import java.text.DateFormat;
15 import java.util.Date;
16
17 public class Hora extends AbstractPageBean
18 {
19    private int __placeholder;
20
21    // método de inicialización de componentes, generado automáticamente.
22    private void _init() throws Exception
23    {
24       // cuerpo vacío
25    } // fin del método _init
26
27    private Page page1 = new Page();
28
29    public Page getPage1()
30    {
```

Figura 26.6 | Archivo de bean de página que establece `textoReloj` a la hora en el servidor Web. (Parte 1 de 4).

```
31          return page1;
32       } // fin del método getPage1
33
34       public void setPage1(Page p)
35       {
36          this.page1 = p;
37       } // fin del método setPage1
38
39       private Html html1 = new Html();
40
41       public Html getHtml1()
42       {
43          return html1;
44       } // fin del método getHtml1
45
46       public void setHtml1(Html h)
47       {
48          this.html1 = h;
49       } // fin del método setHtml1
50
51       private Head head1 = new Head();
52
53       public Head getHead1()
54       {
55          return head1;
56       } // fin del método getHead1
57
58       public void setHead1(Head h)
59       {
60          this.head1 = h;
61       } // fin del método setHead1
62
63       private Link link1 = new Link();
64
65       public Link getLink1()
66       {
67          return link1;
68       } // fin del método getLink1
69
70       public void setLink1(Link l)
71       {
72          this.link1 = l;
73       } // fin del método setLink1
74
75       private Body body1 = new Body();
76
77       public Body getBody1()
78       {
79          return body1;
80       } // fin del método getBody1
81
82       public void setBody1(Body b)
83       {
84          this.body1 = b;
85       } // fin del método setBody1
86
87       private Form form1 = new Form();
88
89       public Form getForm1()
```

Figura 26.6 | Archivo de bean de página que establece `textoReloj` a la hora en el servidor Web. (Parte 2 de 4).

```
90      {
91         return form1;
92      } // fin del método getForm1
93
94      public void setForm1(Form f)
95      {
96         this.form1 = f;
97      } // fin del método setForm1
98
99      private StaticText encabezadoHora = new StaticText();
100
101     public StaticText getEncabezadoHora()
102     {
103        return encabezadoHora;
104     } // fin del método getEncabezadoHora
105
106     public void setEncabezadoHora(StaticText st)
107     {
108        this.encabezadoHora = st;
109     } // fin del método setEncabezadoHora
110
111     private StaticText textoReloj = new StaticText();
112
113     public StaticText getTextoReloj()
114     {
115        return textoReloj;
116     } // fin del método getTextoReloj
117
118     public void setTextoReloj(StaticText st)
119     {
120        this.textoReloj = st;
121     } // fin del método setTextoReloj
122
123     // Construye una nueva instancia de bean de página
124        public Hora()
125     {
126        // constructor vacío
127     } // fin del constructor
128
129     // Devuelve una referencia al bean de datos con ámbito
130     protected RequestBean1 getRequestBean1()
131     {
132        return (RequestBean1)getBean("RequestBean1");
133     } // fin del método getRequestBean1
134
135     // Devuelve una referencia al bean de datos con ámbito
136     protected ApplicationBean1 getApplicationBean1()
137     {
138        return (ApplicationBean1)getBean("ApplicationBean1");
139     } // fin del método getApplicationBean1
140
141     // Devuelve una referencia al bean de datos con ámbito
142     protected SessionBean1 getSessionBean1()
143     {
144        return (SessionBean1)getBean("SessionBean1");
145     } // fin del método getSessionBean1
146
147     // inicializa el contenido de la página
148     public void init()
```

Figura 26.6 | Archivo de bean de página que establece `textoReloj` a la hora en el servidor Web. (Parte 3 de 4).

```
149     {
150        super.init();
151        try
152        {
153           _init();
154        } // fin de try
155        catch ( Exception e )
156        {
157           log( "Error al inicializar Hora", e );
158           throw e instanceof FacesException ? ( FacesException ) e:
159              new FacesException( e );
160        } // fin de catch
161     } // fin del método init
162
163     // método que se llama cuando ocurre una petición de devolución de envío
164     public void preprocess()
165     {
166        // cuerpo vacío
167     } // fin del método preprocess
168
169     // método al que se llama antes de desplegar la página
170     public void prerender()
171     {
172        textoReloj.setValue( DateFormat.getTimeInstance(
173           DateFormat.LONG ).format( new Date() ) );
174     } // fin del método prerender
175
176     // método al que se llama una vez que se completa el despliegue, si se llamó a init
177     public void destroy()
178     {
179        // cuerpo vacío
180     } // fin del método destroy
181  } // fin de la clase Hora
```

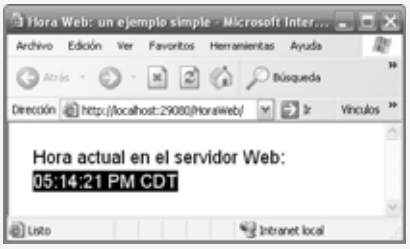

Figura 26.6 | Archivo de bean de página que establece `textoReloj` a la hora en el servidor Web. (Parte 4 de 4).

Este archivo de bean de página proporciona métodos *obtener (get)* y *establecer (set)* para cada elemento del archivo JSP de la figura 26.4. El IDE genera estos métodos de manera automática. Incluimos el archivo de bean de página completo en este primer ejemplo, pero en los siguientes ejemplos omitiremos estas propiedades y sus métodos *obtener* y *establecer* para ahorrar espacio. En las líneas 99 a 109 y 111 a 121 del archivo de bean de página se definen los dos componentes **Static Text** que soltamos en la página, junto con sus métodos *obtener* y *establecer*. Estos componentes son objetos de la clase `StaticText` en el paquete `com.sun.rave.web.ui.component`.

La única lógica requerida en esta página es establecer el texto del componente `textoReloj` para que lea la hora actual en el servidor. Esto lo hacemos en el método `prerender` (líneas 170 a 174). Más adelante hablaremos sobre el significado de éste y otros métodos de bean de página. En las líneas 172 y 173 se obtiene y da formato a la hora en el servidor, y se establece el valor de `textoReloj` con esa hora.