



Criando documentação em HTML com *javadoc* (no CD)

Objetivos

- Apresentar a ferramenta **javadoc** do J2SDK.
- Apresentar comentários de documentação.
- Entender as marcas **javadoc**.
- Ser capaz de gerar documentação de API em HTML com **javadoc**.
- Entender arquivos de documentação gerados pelo **javadoc**.

Oh I get by with a little help from my friends.

John Lennon e Paul McCartney

I feel

The link of nature draw me.

John Milton

I think I shall never see

A poem lovely as a tree.

Joyce Kilmer

Existe apenas uma religião, embora haja uma centena de versões dela.

George Bernard Shaw

O que eu gosto em um bom autor não é o que ele diz, mas o que ele sussurra.

Logan Pearsall Smith

Voltarei.

Douglas MacArthur



Sumário do apêndice

- F.1 Introdução
- F.2 Comentários de documentação
- F.3 Documentando o código-fonte Java
- F.4 javadoc
- F.5 Arquivos produzidos por javadoc

Terminologia

F.1 Introdução

Neste apêndice, fornecemos uma introdução ao utilitário **javadoc** do Java 2 Software Development Kit para criar arquivos de HTML que documentam o código em Java. Esta é a ferramenta utilizada pela Sun Microsystems para criar a documentação da API Java (Fig. F.1.) Discutimos os comentários especiais de Java e as marcas exigidas por **javadoc** para criar documentação baseada em seu código-fonte e como executar a ferramenta **javadoc**.

Para obter informações detalhadas sobre **javadoc**, visite a *homepage* de **javadoc** em

`java.sun.com/j2se/1.3/docs/tooldocs/javadoc/index.html`

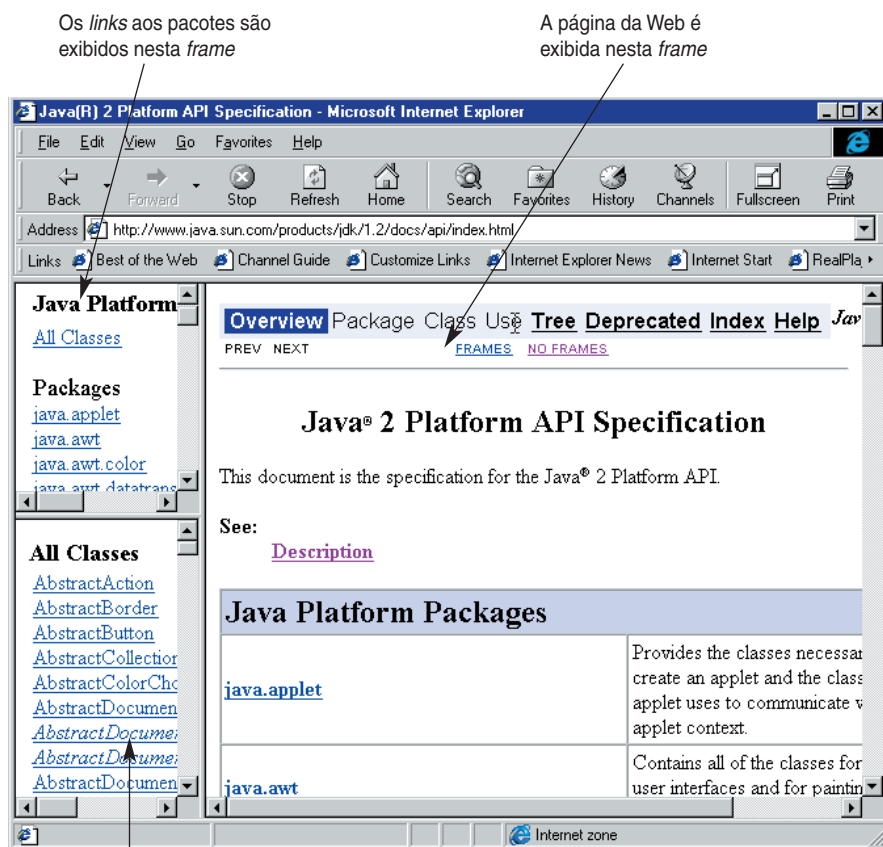


Fig. F.1 A documentação da API Java.

F.2 Comentários de documentação

Antes que os arquivos em HTML possam ser gerados com a ferramenta `javadoc`, os programadores devem inserir comentários especiais – chamados de *comentários de documentação* — em seus arquivos-fonte. Os comentários de documentação são os únicos comentários reconhecidos por `javadoc`. Os comentários de documentação iniciam com `/**` e terminam com `*/`. Um exemplo de um comentário simples de documentação é

```
/** Ordena array de inteiros com o algoritmo MySort */
```

Da mesma forma que outros comentários, os comentários de documentação não são traduzidos em *bytecodes*. Como `javadoc` é utilizado para criar arquivos de HTML, os comentários de documentação podem conter marcas de HTML. Por exemplo, o comentário de documentação

```
/** Ordena array de inteiros com o algoritmo <em>MySort</em> */
```

que contém as marcas de ênfase de HTML, `` e ``, é válido. Nos arquivos de HTML gerados, `MySort` aparecerá como texto enfatizado (normalmente em itálico). Como veremos, as marcas de `javadoc` podem ser inseridas nos comentários de documentação para ajudar o `javadoc` a documentar seu código fonte. Essas marcas – que iniciam com o símbolo `@` – não são marcas de HTML.

F.3 Documentando o código-fonte Java

A Fig. F.2 apresenta uma versão modificada da classe `Time3` da Fig. 8.8 que contém comentários de documentação. No texto que segue o exemplo, discutimos completamente cada uma das marcas de `javadoc` utilizadas nos

```

1  // Fig. F.2: Time3.java
2  // Definição da classe Time3 com os métodos set e get
3  package com.deitel.jhttp4.appenF;
4
5  // Java core packages
6  import java.text.DecimalFormat;
7
8  /**
9   * This class maintains the time in 24-hour format.
10  * @see java.lang.Object
11  * @author Deitel & Associates, Inc.
12  */
13  public class Time3 extends Object {
14
15      private int hour;    // 0 - 23
16      private int minute; // 0 - 59
17      private int second; // 0 - 59
18
19
20      /**
21       * Time3 constructor initializes each instance variable
22       * to zero. Ensures that Time object starts in a
23       * consistent state.
24       * @throws <code>Exception</code> in the case of an invalid time
25       */
26      public Time3() throws Exception
27      {
28          setTime( 0, 0, 0 );
29      }
30
31
32      /**
33       * Time3 constructor: hour supplied, minute and second
34       * defaulted to 0

```

Fig. F.2 Um arquivo de código-fonte em Java contendo comentários de documentação (parte 1 de 4).

```

35     * @param h the hour
36     * @throws <code>Exception</code> in the case of an invalid time
37     */
38     public Time3( int h ) throws Exception
39     {
40         setTime( h, 0, 0 );
41     }
42
43     /**
44     * Time3 constructor: hour and minute supplied, second
45     * defaulted to 0
46     * @param h the hour
47     * @param m the minute
48     * @throws <code>Exception</code> in the case of an invalid time
49     */
50     public Time3( int h, int m ) throws Exception
51     {
52         setTime( h, m, 0 );
53     }
54
55     /**
56     * Time3 constructor: hour, minute and second supplied
57     * @param h the hour
58     * @param m the minute
59     * @param s the second
60     * @throws <code>Exception</code> in the case of an invalid time
61     */
62     public Time3( int h, int m, int s ) throws Exception
63     {
64         setTime( h, m, s );
65     }
66
67     /**
68     * Time3 constructor: another Time3 object supplied
69     * @param time Time3 object
70     * @throws <code>Exception</code> in the case of an invalid time
71     */
72     public Time3( Time3 time ) throws Exception
73     {
74         setTime( time.getHour(), time.getMinute(),
75                 time.getSecond() );
76     }
77
78     // Set Methods
79     /**
80     * Set a new time value using universal time. Perform
81     * validity checks on data. Set invalid values to zero.
82     * @param h the hour
83     * @param m the minute
84     * @param s the second
85     * @see com.deitel.jhttp4.appenF.Time3#setHour
86     * @see Time3#setMinute
87     * @see #setSecond
88     * @throws <code>Exception</code> in the case of an invalid time
89     */
90     public void setTime( int h, int m, int s ) throws Exception
91     {
92         setHour( h ); // set the hour
93         setMinute( m ); // set the minute
94         setSecond( s ); // set the second

```

Fig. F.2 Um arquivo de código-fonte em Java contendo comentários de documentação (parte 2 de 4).

```

95     }
96
97     /**
98     * Sets the hour
99     * @param h the hour
100    * @throws Exception in the case of an invalid time
101    */
102    public void setHour( int h ) throws Exception
103    {
104        if ( h >= 0 && h < 24 )
105            hour = h;
106        else
107            throw new Exception();
108    }
109
110    /**
111    * Sets the minute
112    * @param m the minute
113    * @throws Exception in the case of an invalid time
114    */
115    public void setMinute( int m ) throws Exception
116    {
117        if ( m >= 0 && h < 60 )
118            minute = m;
119        else
120            throw new Exception();
121    }
122
123    /**
124    * Sets the second
125    * @param m the minute
126    * @throws Exception in the case of an invalid time
127    */
128    public void setSecond( int s ) throws Exception
129    {
130        if ( s >= 0 && s < 60 )
131            second = s;
132        else
133            throw new Exception();
134    }
135
136    // Get Methods
137    /**
138    * Gets the hour
139    * @return an <code>int</code> specifying the hour.
140    */
141    public int getHour()
142    {
143        return hour;
144    }
145
146    /**
147    * Gets the minute
148    * @return an <code>int</code> specifying the minute.
149    */
150    public int getMinute()
151    {
152        return minute;
153    }
154

```

Fig. F.2 Um arquivo de código-fonte em Java contendo comentários de documentação (parte 3 de 4).

```

155     /**
156     * Gets the second
157     * @return an <code>int</code> specifying the second.
158     */
159     public int getSecond()
160     {
161         return second;
162     }
163
164     /**
165     * Convert to <code>String</code> in universal-time format
166     * @return a <code>String</code> representation
167     * of the time in universal-time format
168     */
169     public String toUniversalString()
170     {
171         DecimalFormat twoDigits = new DecimalFormat( "00" );
172
173         return twoDigits.format( getHour() ) + ":" +
174             twoDigits.format( getMinute() ) + ":" +
175             twoDigits.format( getSecond() );
176     }
177
178     /**
179     * Convert to <code>String</code> in standard-time format
180     * @return a <code>String</code> representation
181     * of the time in standard-time format
182     */
183     public String toString()
184     {
185         DecimalFormat twoDigits = new DecimalFormat( "00" );
186
187         return ( ( getHour() == 12 || getHour() == 0 ) ?
188             12 : getHour() % 12 ) + ":" +
189             twoDigits.format( getMinute() ) + ":" +
190             twoDigits.format( getSecond() ) +
191             ( getHour() < 12 ? " AM" : " PM" );
192     }
193 }

```

Fig. F.2 Um arquivo de código-fonte em Java contendo comentários de documentação (parte 4 de 4).

comentários de documentação. Discutimos como utilizar a ferramenta **javadoc** para gerar documentação em HTML a partir desse arquivo na Seção F.4.

Os comentários de documentação são colocados na linha anterior a uma definição de classe, uma definição de interface, um construtor, um método e um campo (isto é, uma variável de instância ou uma referência). O primeiro comentário de documentação (linhas 8 a 12) apresenta a classe **Time3**. A linha

```
* This class maintains the time in 24-hour format.
```

é uma descrição da classe **Time3** fornecida pelo programador. A descrição pode conter quantas linhas forem necessárias para fornecer uma descrição da classe para qualquer programador que venha a utilizá-la. As marcas **@see** e **@author** são utilizados para especificar uma nota **See Also:** e uma nota **Author:**, respectivamente, na documentação em HTML (Fig. F.3). A nota **See Also:** especifica outras classes relacionadas que podem ser de interesse para o programador que utiliza essa classe. A marca **@author** especifica o autor da classe. Pode-se usar mais de uma marca **@author** para documentar múltiplos autores. Observe que os asteriscos (*) em cada linha entre **/**** e ***/** não são necessários. Essa é uma convenção utilizada por programadores para alinhar descrições e marcas de **java-doc**. Ao analisar sintaticamente um comentário de documentação, **javadoc** descarta todos os caracteres de espaço em branco até o primeiro caractere diferente de espaço em branco em cada linha. Se o primeiro caractere diferente de espaço em branco encontrado for um asterisco, ele também é descartado.

Repare que esse comentário de documentação precede imediatamente a definição de classe – qualquer código colocado entre o comentário de documentação e a definição de classe faz com que o **javadoc** ignore o comentário de documentação. Isso também é verdadeiro para outras estruturas de código (por exemplo, construtores, métodos, variáveis de instância, etc.)



Erro comum de programação F.1

Colocar uma instrução **import** entre o comentário de classe e a declaração de classe é um erro de lógica. Isso faz com que o comentário de classe seja ignorado pelo **javadoc**.

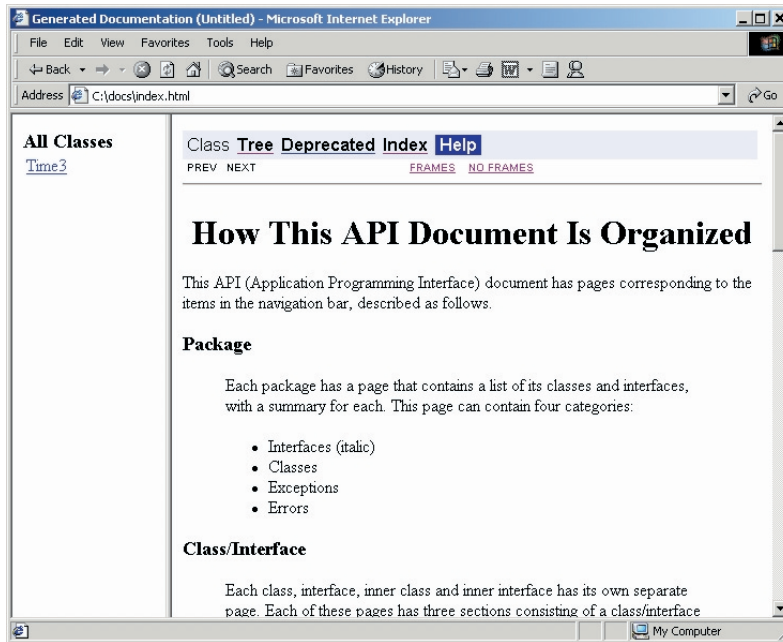


Fig. F.3 A documentação em HTML para a classe **Time3**.



Observação de engenharia de software F.1

Definir vários campos em uma instrução separada por vírgulas com um único comentário acima dessa instrução resultará na utilização desse comentário pelo **javadoc** para todos os campos.

O comentário de documentação nas linhas 32 a 37 descreve um dos construtores de **Time3**. A marca **@param** descreve um parâmetro para o método. Os parâmetros aparecem no documento de HTML em uma nota **Parameters:** (Fig. F.4), que é seguida por uma lista de todos os parâmetros especificados com a marca **@param**. Para esse construtor, o nome do parâmetro é **h** e sua descrição é “the hour”. Pode-se usar a marca **@param** somente com métodos e construtores.

A marca **@throws** especifica as exceções disparadas por esse método. Como as marcas **@param**, as marcas **@throws** são utilizadas somente com métodos e construtores. Deve-se fornecer uma **@throws** para cada tipo de exceção disparada pelo método.



Observação de engenharia de software F.2

Para produzir documentação **javadoc** adequada, você deve declarar cada variável de instância em uma linha separada.

Os comentários de documentação podem conter várias marcas `@param` e `@see`. O comentário de documentação nas linhas 79 a 89 descreve o método `setTime`. O HTML gerado para esse método é mostrado na Fig. F.5. Três marcas `@param` descrevem os parâmetros do método. Isso resulta em uma nota **Parameters:** que lista os três parâmetros. Os métodos `setHour`, `setMinute` e `setSecond` são marcados com `@see` para criar *hyperlinks* para suas descrições no documento HTML. Usa-se um caractere `#` em vez de um ponto ao se marcar um método ou um campo. Isso cria um *link* para o nome que segue o caractere `#`. Demonstramos três maneiras diferentes para marcar métodos (isto é, o nome completamente qualificado, a qualificação do nome de classe e nenhuma qualificação) utilizando `@see` nas linhas 85 a 87. Se o nome completamente qualificado não for dado (como nas linhas 86 e 87), **javadoc** procura o método ou o campo especificado na seguinte ordem: na classe atual, nas superclasses, no pacote e nos arquivos importados.

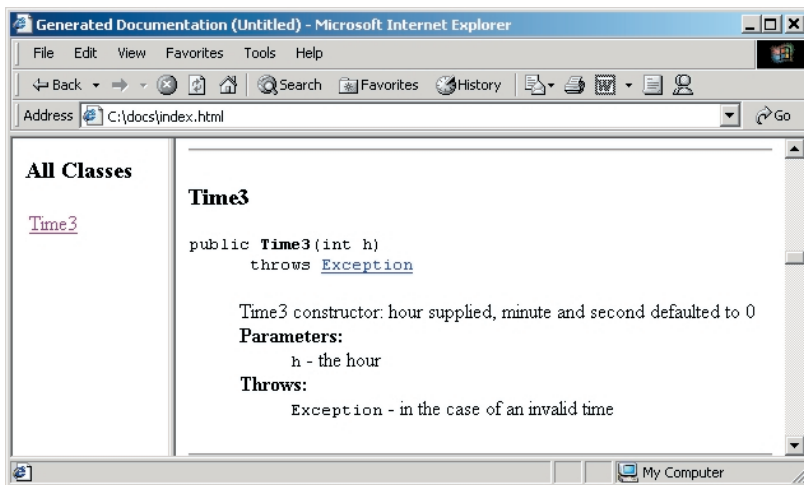


Fig. F.4 A nota **Parameters:** gerada por **javadoc**.

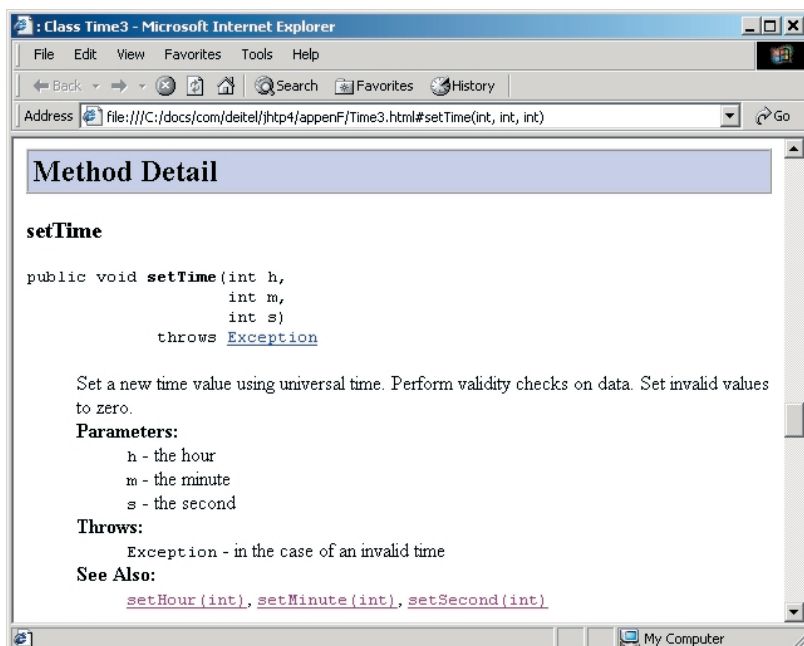


Fig. F.5 A documentação em HTML para o método `setTime`.

A única outra marca utilizada neste arquivo é **@return**, que especifica uma nota **Returns:** na documentação em HTML (Fig F.6.) O comentário nas linhas 137 a 140 documenta o método **getHour**. A marca **@return** descreve um tipo de retorno do método para ajudar o programador a entender como utilizar o valor devolvido pelo método. Pela convenção do **javadoc**, os programadores compõem o código-fonte (isto é, palavras-chave, identificadores, expressões, etc.) com as marcas de HTML `<code>` e `</code>`.

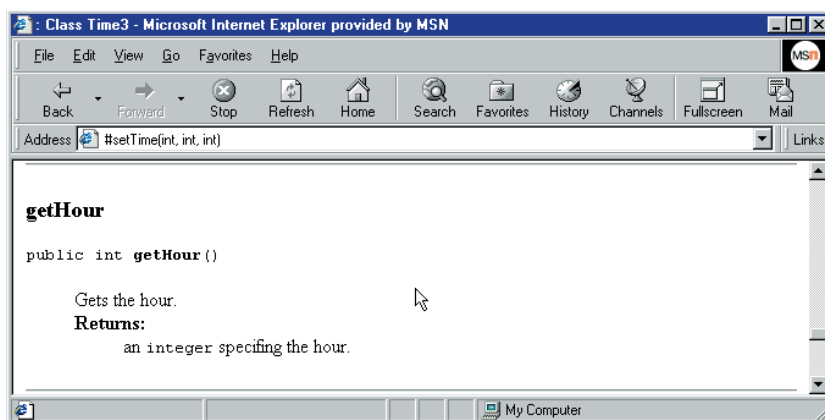


Fig. F.6 A documentação em HTML para **getHour**.



Boa prática de programação F.1

Mudar as fontes do código-fonte para marcas de **javadoc** ajuda a destacar os nomes de código do resto da descrição.

Além das marcas apresentadas nesse exemplo, **javadoc** reconhece seis outras marcas. A Fig. F.7 resume as marcas mais usadas.

Marca javadoc	Descrição
@author	Adiciona uma nota Author: se a opção -author é usada ao executar javadoc .
@param	Usada para descrever os parâmetros de métodos e construtores.
@return	Adiciona uma nota Return: que descreve o tipo de valor devolvido por um método.
@see	Adiciona uma nota See Also: que contém <i>hyperlinks</i> para classes ou métodos relacionados.
@throws	Adiciona uma nota Throws: que especifica as exceções disparadas pelo método. @exception é um sinônimo para @throws .
@deprecated	Adiciona uma nota Deprecated . Estas são notas para programadores, que indicam que eles não devem utilizar os recursos especificados da classe. As notas Deprecated normalmente aparecem quando uma classe foi melhorada com recursos novos e melhores, mas recursos mais velhos são mantidos para compatibilidade com versões anteriores.
@link	Permite que programador insira um <i>hyperlink</i> explícito para outro documento HTML.
@since	Adiciona uma nota Since . Essas notas são utilizadas para novas versões de uma classe para indicar quando um recurso foi introduzido pela primeira vez. Por exemplo, a documentação da Java API utiliza isso para indicar os recursos que foram introduzidos em Java 1.0, Java 1.1 e Java 2.
@version	Adiciona uma nota Version . Essas notas ajudam a manter o número de versão do <i>software</i> que contém a classe ou o método.

Fig. F.7 Marcas comuns de **javadoc**.

F.4 javadoc

Nesta seção, discutimos como aplicar a ferramenta **javadoc** sobre um arquivo fonte Java, para criar documentação em HTML para a classe no arquivo. Como outras ferramentas, **javadoc** é executado a partir da linha de comando. A forma geral do comando **javadoc** é

```
javadoc opções pacotes fontes @arquivos
```

onde *opções* é uma lista das *opções* de linhas de comando, *pacotes* é uma lista de pacotes que o usuário gostaria de documentar, *fontes* é uma lista de arquivos-fonte de Java a documentar e *@arquivos* é um arquivo de texto que contém os nomes dos pacotes e/ou arquivos-fonte a enviar para o utilitário **javadoc**, para que você possa criar documentação para aqueles pacotes e classes. O caractere curinga ***** pode ser usado para especificar várias origens (por exemplo, **c:*.java**). [Nota: todos os itens são separados por espaços e *@arquivos* é uma palavra.] A Fig. F. 8 mostra uma janela DOS que contém o comando **javadoc** que digitamos para gerar a documentação em HTML.

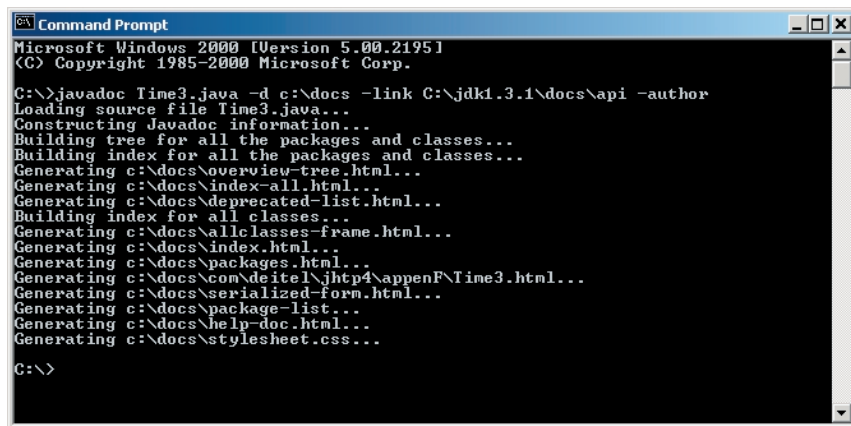
Na Fig. F.8, o argumento **-d** especifica o diretório (por exemplo, **c:\docs**) em que os arquivos HTML serão armazenados em disco. Utilizamos a opção **-link** para que nossa documentação seja vinculada com a documentação da Sun (instalada em nossa unidade **E:**). Isso cria um *hyperlink* entre nossa documentação e a documentação da Sun (veja a Fig. F.5, em que a classe de Java **Exception** do pacote **java.lang** é vinculada com um *hyperlink*). Sem o argumento **-link**, **Exception** aparece como texto no documento de HTML – não um *hyperlink*. O argumento **-author** instrui **javadoc** a processar a marca **@author** (por *default*, ele ignora essa marca).

F.5 Arquivos produzidos por javadoc

Na última seção, aplicamos a ferramenta **javadoc** no arquivo **Time3.java**. Quando **javadoc** é executado, ele exibe o nome de cada arquivo HTML que cria (veja a Fig. F.8). A partir do arquivo-fonte, **javadoc** criou um documento HTML para a classe, denominado **Time3.html**. Se o arquivo-fonte contiver múltiplas classes ou interfaces, cria-se um documento HTML separado para cada classe. Como a classe **Time3** pertence a um pacote, a página é criada no diretório **C:\docs\com\deitel\jhtp4\appenF** (em plataformas Win32). O diretório **c:\docs** foi especificado com a opção de linha de comando **-d** do **javadoc** e os diretórios restantes foram criados com base na instrução **package**.

Outro arquivo que **javadoc** cria é **index.html**, a página inicial de HTML na documentação. Para visualizar a documentação que você gera com o **javadoc**, carregue **index.html** em seu navegador da Web. Na Fig. F.9, a *frame* direita contém a página **index.html** e a esquerda contém a página **allclasses-frame.html**, que contém *links* para as classes do código-fonte. [Nota: uma vez que nosso exemplo não contém múltiplos pacotes, não há nenhuma *frame* que lista os pacotes. Normalmente essa *frame* apareceria acima da *frame* esquerda (contendo “All Classes”), como na Fig. F.1.]

A *barra de navegação* (na parte superior da *frame* direita na Fig. F.9) indica qual a página HTML que está atualmente carregada, destacando o *link* da página (por exemplo, o *link* **Class** na Fig. F.9).



```

Microsoft Windows [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\>javadoc Time3.java -d c:\docs -link C:\jdk1.3.1\docs\api -author
Loading source file Time3.java...
Constructing Javadoc information...
Building tree for all the packages and classes...
Building index for all the packages and classes...
Generating c:\docs\overview-tree.html...
Generating c:\docs\index-all.html...
Generating c:\docs\deprecated-list.html...
Building index for all classes...
Generating c:\docs\allclasses-frame.html...
Generating c:\docs\index.html...
Generating c:\docs\packages.html...
Generating c:\docs\com\deitel\jhtp4\appenF\Time3.html...
Generating c:\docs\serialized-form.html...
Generating c:\docs\package-list.html...
Generating c:\docs\help-doc.html...
Generating c:\docs\stylesheet.css...

C:\>

```

Fig. F.8 Utilizando a ferramenta **javadoc**.

Clicar no link **Tree** (Fig. F.10) exibe uma hierarquia de classes para todas as classes exibidas na *frame* esquerda. Em nosso exemplo, documentamos somente a classe **Time3** – que estende **Object**. Clicar no link **Deprecated** carrega **deprecated-list.html** na *frame* direita. Essa página contém uma lista de todos os nomes obsoletos. Uma vez que não utilizamos a marca **@deprecated** nesse exemplo, essa página não contém nenhuma informação. Clicar no link **Index** carrega a página **index-all.html**, a qual contém uma lista em ordem alfabética de todas as classes, interfaces, métodos e campos.

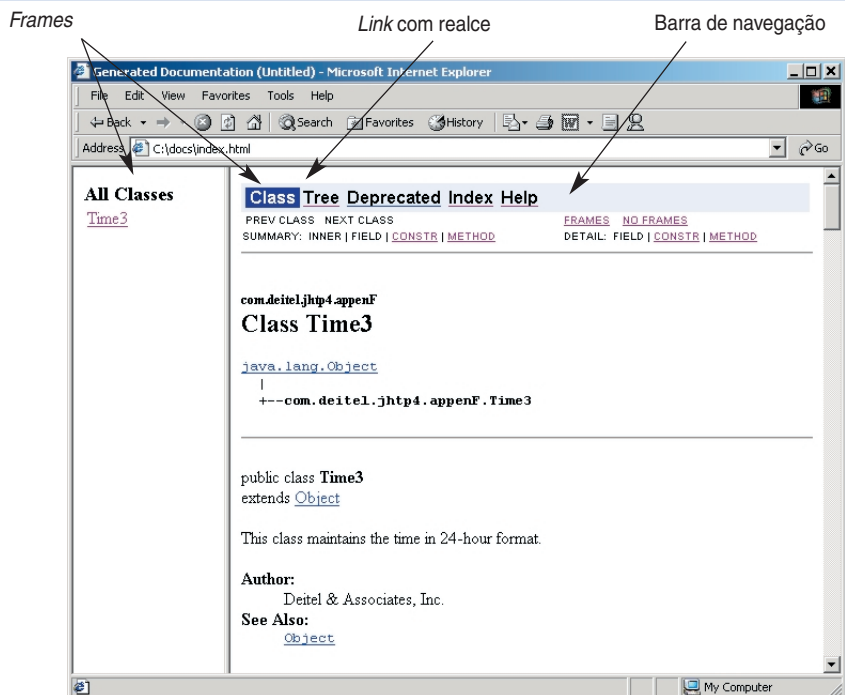


Fig. F.9 index.html da classe Time3.

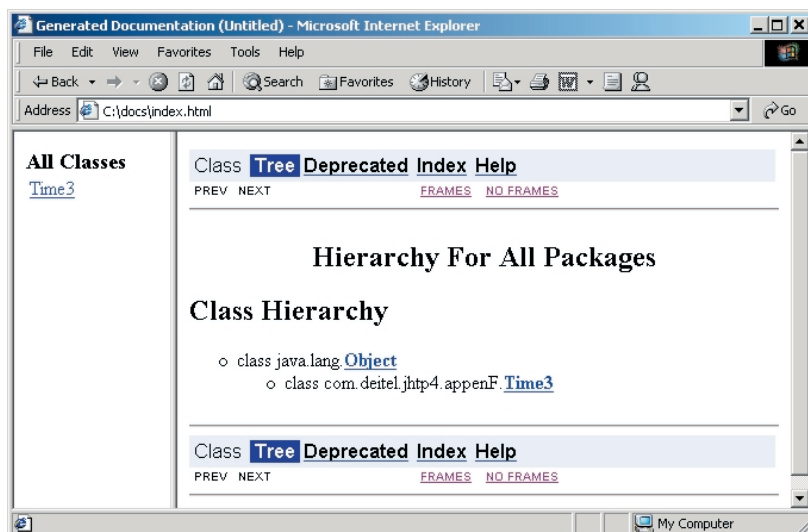


Fig. F.10 A página Tree.

A Fig. F.11 a mostra a página `index-all.html` da classe `Time3` carregada em um navegador da Web. Clicar no link **Help** carrega `helpdoc.html` (Fig. F.12). Este é um arquivo de ajuda para navegar pela documentação. Fornece-se um arquivo de ajuda *default*, mas o programador pode especificar outros arquivos de ajuda.

Entre os outros arquivos gerados por `javadoc` estão `serialized-form.html`, que documenta as classes `Serializable` e `Externalizable`, e `package-list.txt`, que é utilizado pelo argumento de linha de comando `-link` e não faz realmente parte da documentação.

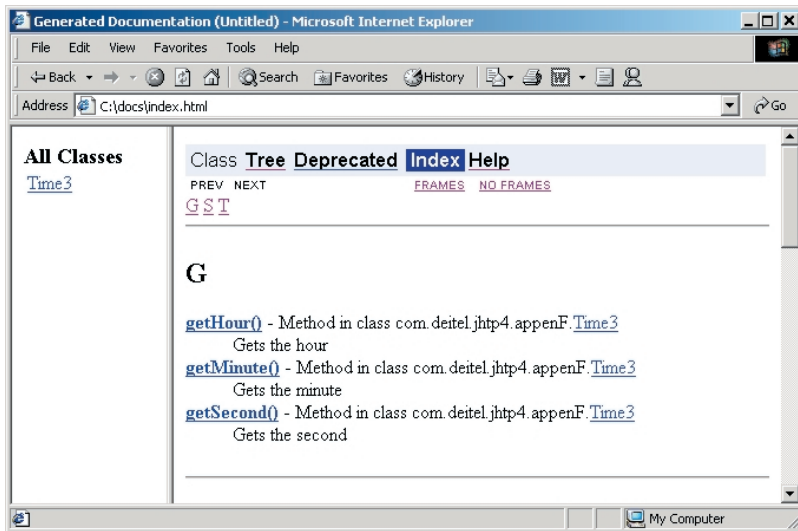


Fig. F.11 A página `index-all.html` de `Time3`.

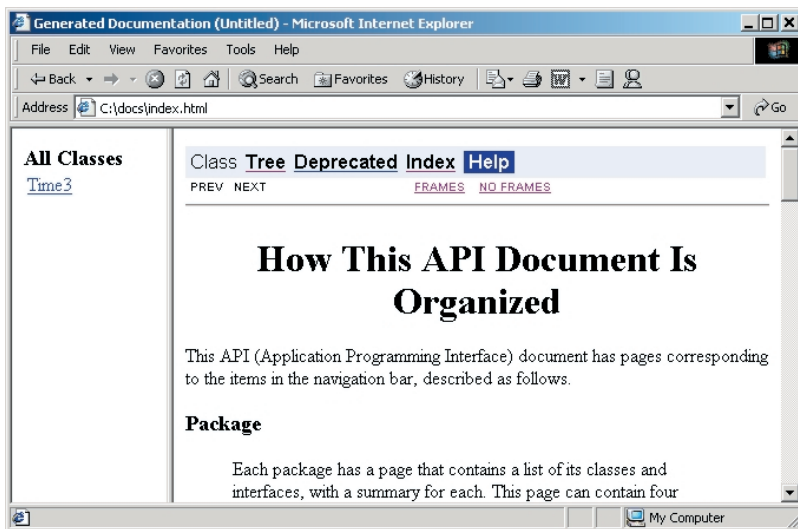


Fig. F.12 A página `helpdoc.html` de `Time3`.

Terminologia

`allclasses-frame.html`

argumento `-author`

argumento `-d`

argumento `-link`

comentário de documentação

`deprecated-list.html`

Help

`helpdoc.html`

`index.html`

`index-all.html`

item **Class** na barra do navegador

`javadoc`

link **Tree**

marca `@author` de `javadoc`

marca `@deprecated`

marca `@exception`

marca `@link`

marca `@param`

marca `@return`

marca `@see`

marca `@serial`

marca `@serialData`

marca `@since`

marca `@throws`

marca `@version`

`name-frame.html`

nota **Exception**

nota **Overrides:**

nota **Parameters:**

nota **Returns:**

nota **See Also:**

`serialized-form.html`