M Criando documentação com o javadoc

M.1 Introdução

Neste apêndice, fornecemos uma introdução a **javadoc** — uma ferramenta utilizada para criar arquivos HTML que documentam o código Java. Essa ferramenta é utilizada pela Sun para criar a documentação da Java API (Figura M.1). Discutimos os comentários especiais do Java e os tags requeridos por `javadoc` para criar documentação baseada em seu código-fonte e como executar a ferramenta `javadoc`. Para informações detalhadas sobre `javadoc`, visite a home page `javadoc` em

```
java.sun.com/javase/6/docs/technotes/guides/javadoc/index.html
```

M.2 Comentários da documentação

Antes que arquivos de HTML possam ser gerados com a ferramenta `javadoc`, os programadores devem inserir comentários especiais — chamados **comentários de documentação** — em seus arquivos de fonte. Os comentários de documentação são os únicos comentários reconhecidos pelo `javadoc`. Comentários de documentação iniciam com **/**** e terminam com ***/**. Como ocorre com comentários tradicionais, os comentários de documentação podem abranger múltiplas linhas. Um exemplo de um comentário simples de documentação é

```
/** Classifica um array de inteiros utilizando o algoritmo MySort */
```

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

```
/** Classifica o array de inteiros utilizando o algoritmo <strong>MySort</strong> */
```

contém os tags HTML em negrito `<strong>` e `</strong>`. Nos arquivos de HTML gerados, `MySort` aparecerá em negrito. Como veremos, os **tags `javadoc`** também podem ser inseridos nos comentários de documentação para ajudar o `javadoc` a documentar seu código-fonte. Esses tags — que começam com um símbolo `@` — não são tags HTML.

M.3 Documentando o código-fonte Java

Nesta seção, documentamos uma versão modificada da classe `Time2` da Figura 8.5 utilizando comentários de documentação. No texto que segue o exemplo, discutimos completamente cada um dos tags `javadoc` utilizados nos comentários de documentação. Na próxima seção, discutiremos como utilizar a ferramenta `javadoc` para gerar documentação em HTML a partir desse arquivo.

```
1  // Figura M.1: Time.java
2  // Declaração da classe Time com os métodos set e get.
3  package com.deitel; // coloca a classe Time em um pacote
4
```

```
5  /**
6   * Essa classe mantém a hora no formato de 24 horas.
7   *  @see java.lang.Object
8   *  @author Deitel & Associates, Inc.
9   */
10 public class Time
11 {
12    private int hour;   // 0 - 23
13    private int minute; // 0 - 59
14    private int second; // 0 - 59
15
16    /**
17     *  O construtor Time sem argumento inicializa cada variável de instância
18     *  como zero. Isso assegura que objetos Time iniciem em um estado
19     *  consistente. @throws Exceção no caso de uma data/hora inválida
20     */
21    public Time() throws Exception
22    {
23       this( 0, 0, 0 ); // invoca o construtor Time com três argumentos
24    } // fim do construtor Time sem argumento
25
26    /**
27     *  Construtor Time
28     *  @param h a hora
29     *  @throws Exceção no caso de uma data/hora inválida
30     */
31    public Time( int h ) throws Exception
32    {
33       this( h, 0, 0 ); // invoca o construtor Time com três argumentos
34    } // fim do construtor Time de um argumento.
35
36    /**
37     *  Construtor Time
38     *  @param h a hora
39     *  @param m o minuto
40     *  @throws Exceção no caso de uma data/hora inválida
41     */
42    public Time( int h, int m ) throws Exception
43    {
44       this( h, m, 0 ); // invoca o construtor Time com três argumentos
45    } // fim do construtor Time de três argumentos
46
47    /**
48     *  Construtor Time
49     *  @param h a hora
50     *  @param m o minuto
51     *  @param s o segundo
52     *  @throws Exceção no caso de uma data/hora inválida
53     */
54    public Time( int h, int m, int s ) throws Exception
55    {
56       setTime( h, m, s ); // invoca setTime para validar a data/hora
57    } // fim do construtor Time de três argumentos
58
59    /**
60     *  constructor Time
61     *  @param time Um objeto Time com o qual inicializar
62     *  @throws Exceção no caso de uma data/hora inválida
63     */
64    public Time( Time time ) throws Exception
65    {
66       // invoca o construtor Time com três argumentos
67       this( time.getHour(), time.getMinute(), time.getSecond() );
68    } // fim do construtor Time com o argumento Time
69
70    /**
71     * Configura um novo valor usando hora universal. Verifica
72     *  a validade dos dados. Configura valores inválidos como zero.
73     *  @param h a hora
```

```
74          *  @param m o minuto
75          *  @param s o segundo
76          *  @see com.deitel.jhtp6.appenH.Time#setHour
77          *  @see Time#setMinute
78          *  @see #setSecond
79          *  @throws Exceção no caso de uma data/hora inválida
80          */
81       public void setTime( int h, int m, int s ) throws Exception
82       {
83          setHour( h );   // configura hour
84          setMinute( m ); // configura minute
85          setSecond( s ); // configura second
86       } // fim do método setTime
87
88       /**
89        *  Configura a hora.
90        *  @param h a hora
91        *  @throws Exceção no caso de uma data/hora inválida
92        */
93       public void setHour( int h ) throws Exception
94       {
95          if ( h >= 0 && h < 24 )
96             hour = h;
97          else
98             throw( new Exception() );
99       } // fim do método setHour
100
101      /**
102       *  Configura o minuto.
103       *  @param m o minuto
104       *  @throws Exceção no caso de uma data/hora inválida
105       */
106      public void setMinute( int m ) throws Exception
107      {
108         if ( m >= 0 && m < 60 )
109            minute = m;
110         else
111            throw( new Exception() );
112      } // fim do método setMinute
113
114      /**
115       *  Configura o segundo.
116       *  @param s o segundo.
117       *  @throws Exceção no caso de uma data/hora inválida
118       */
119      public void setSecond( int s ) throws Exception
120      {
121         if ( s >= 0 && s < 60 )
122            second = s;
123         else
124            throw( new Exception() );
125      } // fim do método setSecond
126
127      /**
128       *  Obtém a hora.
129       *  @return um <code>integer</code> especificando a hora.
130       */
131      public int getHour()
132      {
133         return hour;
134      } // fim do método getHour
135
136      /**
137       *  Obtém o minuto.
138       *  @return um <code>integer</code> especificando o minuto.
139       */
140      public int getMinute()
141      {
142         return minute;
```

```
143       } // fim do método getMinute
144
145       /**
146        *  Obtém o segundo.
147        *  @return um <code>integer</code> especificando o segundo.
148        */
149       public int getSecond()
150       {
151          return second;
152       } // fim do método getSecond
153
154       /**
155         // Converte para String no formato de data/hora universal
156        *  @return uma representação de <code>string<code>
157        *  da data/hora no formato de data/hora universal
158        */
159       public String toUniversalString()
160       {
161          return String.format(
162             "%02d:%02d:%02d", getHour(), getMinute(), getSecond() );
163       } // fim do método do toUniversalString
164
165       /**
166        *  Converte para String no formato de data/hora padrão
167        *  @return uma representação de <code>string<code>
168        *  da data/hora no formato padrão de data/hora
169        */
170       public String toStandardString()
171       {
172          return String.format( "%d:%02d:%02d %s",
173             ( ( getHour() == 0 || getHour() == 12 ) ? 12 : getHour() % 12 ),
174             getMinute(), getSecond(), ( getHour() < 12 ? "AM" : "PM" ) );
175       } // fim do método toStandardString
176    } // fim da classe Time
```

Figura M.1 | Arquivo de código-fonte Java contendo comentários de documentação.

Os comentários de documentação são colocados na linha antes de uma declaração de classe, uma declaração de interface, um construtor, um método e um campo (isto é, uma variável de instância ou uma referência). O primeiro comentário de documentação (linhas 5–9) introduz a classe `Time`. A linha 6 é uma descrição da classe `Time` fornecida pelo programador. A descrição pode conter quantas linhas forem necessárias para fornecer uma descrição da classe para que qualquer programador possa utilizar. Os tags **`@see`** e **`@author`** são utilizados para especificar uma nota **See Also:** e uma nota **Author:**, respectivamente na documentação em HTML (Figura M.2). A nota See Also: especifica outras classes relacionadas que podem ser de interesse para um programador que utiliza essa classe. O tag `@author` especifica o autor da classe. Mais de um tag `@author` pode ser utilizado para documentar múltiplos autores. [*Nota*: Os asteriscos (`*`) em cada linha entre `/**` e `*/` não são exigidos. Entretanto, essa é a convenção recomendável para alinhar descrições e tags `javadoc`. Ao analisar sintaticamente um comentário de documentação, o `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 de espaço não branco encontrado for um asterisco, ele também é descartado.]

Observe que esse comentário de documentação precede imediatamente a declaração de classe — qualquer código colocado entre o comentário de documentação e a declaração de classe faz com que o `javadoc` ignore o comentário da documentação. Isso também é verdadeiro para outras estruturas de código (por exemplo, construtores, métodos, variáveis de instância).

Erro de programação comum M.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`.

Observação de engenharia de software

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.

Observação de engenharia de software

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

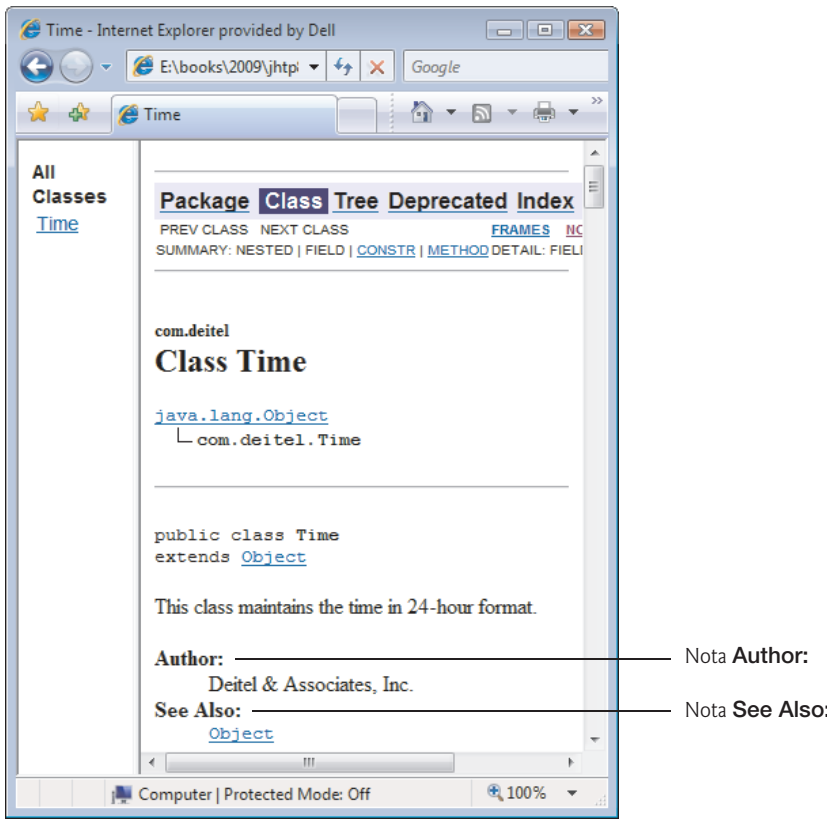


Figura M.2 | Notas **Author:** e **See Also:** geradas pelo `javadoc`.

O comentário de documentação nas linhas 26–30 descreve o construtor `Time`. O tag **`@param`** descreve um parâmetro para o construtor. Os parâmetros aparecem no documento HTML em uma nota **Parameters:** (Figura M.3) que é seguida por uma lista de todos os parâmetros especificados com o tag `@param`. Para esse construtor, o nome do parâmetro é `h` e sua descrição é `"the hour"`. O tag `@param` pode ser utilizado somente com métodos e construtores.

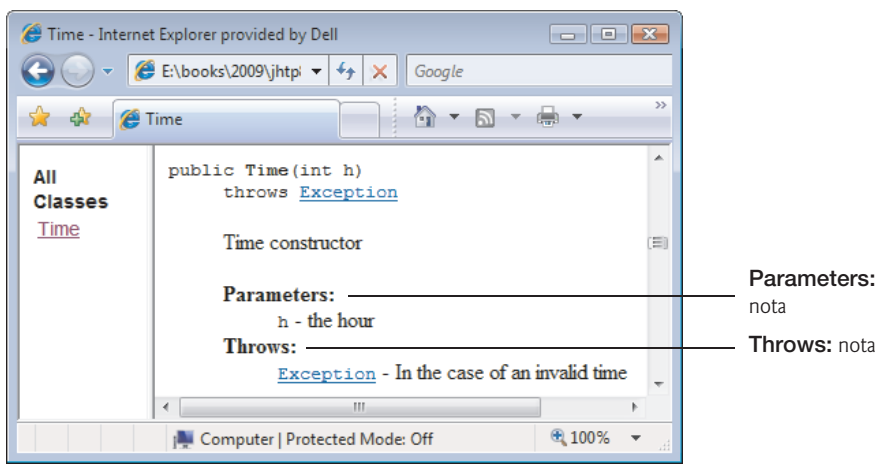


Figura M.3 | Notas **Parameters:** e **Throws:** geradas por `javadoc`.

O tag **`@throws`** especifica as exceções lançadas por esse construtor. Como as tags `@param`, as tag `@throws` somente são utilizadas com métodos e construtores. Um `@throws` deve ser fornecido para cada tipo de exceção lançado pelo método.

Os comentários de documentação podem conter múltiplos tags `@param` e `@see`. O comentário de documentação nas linhas 70–80 descreve o método `setTime`. A HTML gerada para esse método é mostrada na Figura M.4. Três tags `@param` descrevem os parâmetros do método.

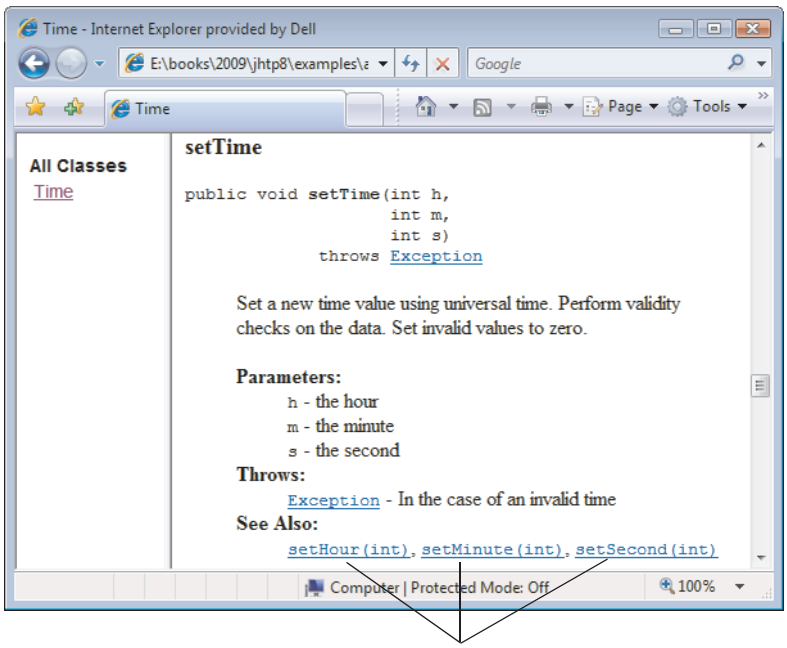


Figura M.4 | A documentação em HTML para o método `setTime`.

Isso resulta em uma nota **Parameters:** que lista os três parâmetros. Os métodos `setHour`, `setMinute` e `setSecond` são rotulados com `@see` para criar hyperlinks para suas descrições no documento de HTML. Um caractere **#** é utilizado em vez de um ponto quando rotular um método ou um campo. Isso cria um link ao nome do campo ou do método que segue o caractere #. Demonstramos três maneiras diferentes (isto é, o nome completamente qualificado, a qualificação do nome de classe e nenhuma qualificação) para marcar métodos utilizando `@see` nas linhas 76–78. A linha 76 utiliza o nome completamente qualificado para marcar o método `setHour`. Se o nome completamente qualificado não for fornecido (linhas 77 e 78), o `javadoc` procura o método ou campo especificado na ordem a seguir: a classe atual, as superclasses, o pacote e os arquivos importados.

O único tag além daquele utilizado nesse arquivo é **`@return`**, que especifica uma nota **Returns:** na documentação em HTML (Fig M.5) O comentário nas linhas 127–130 documenta o método `getHour`. O tag `@return` descreve um tipo de retorno do método para ajudar o programador a entender como utilizar o valor de retorno do método. Pela convenção do `javadoc`, os programadores compõem o código-fonte (isto é, palavras-chave, identificadores e expressões) com os tags de HTML `<code>` e `</code>`. Vários outros tags `javadoc` são brevemente resumidos na Figura M.6.

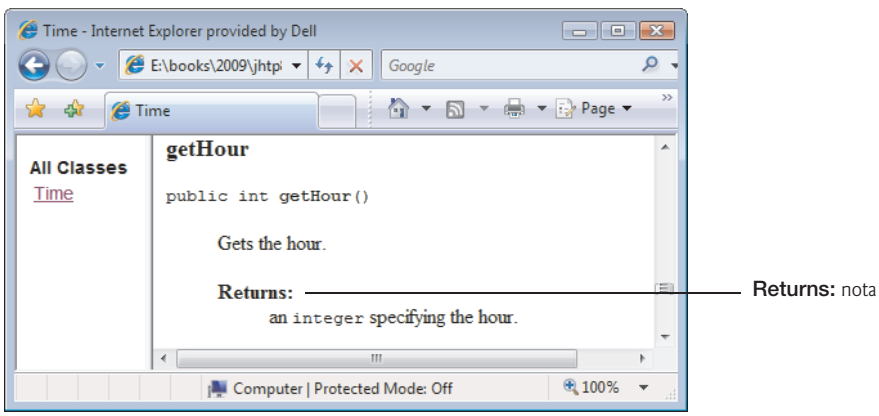


Figura M.5 | A documentação em HTML para o método `getHour`.

Boa prática de programação M.1

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

Tag javadoc	Descrição
`@deprecated`	Adiciona uma nota **Deprecated**. Essas são notas para os programadores indicando que eles não devem utilizar os recursos especificados da classe. Notas `Deprecated` normalmente aparecem quando uma classe foi aprimorada com novos e melhores recursos e os recursos mais antigos são mantidos para retrocompatibilidade.
`{@link}`	Isso permite que o programador insira um hyperlink explícito em outro documento de HTML.
`@since`	Adiciona uma nota Since:. Essas notas são utilizadas para novas versões de uma classe para indicar quando um recurso foi introduzido primeiro. Por exemplo, a documentação da Java API utiliza isso para indicar os recursos que foram introduzidos no Java 1.5.
`@version`	Adiciona uma nota Version. Essas notas ajudam a manter o número de versão do software contendo a classe ou o método.

Figura M.6 | Outros tags `javadoc` comuns.

M.4 javadoc

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

`javadoc` *options packages sources @files*

onde *options* é uma lista de opções de linha de comando, *packages* é uma lista de pacotes que o usuário gostaria de documentar, *sources* é uma lista de arquivos-fonte Java para documentar e *@files* é uma lista de arquivos de texto contendo as opções `javadoc`, os nomes de pacotes e/ou arquivos-fonte para enviar ao utilitário `javadoc`. [*Nota*: Todos itens são separados por espaços e *@files* é uma palavra.] A Figura M.7 mostra uma janela Command Prompt contendo o comando `javadoc` que digitamos para gerar a documentação em HTML. Para informações detalhadas sobre o comando `javadoc`, visite o guia de referência `javadoc` e exemplos em `java.sun.com/j2se/5.0/docs/tooldocs/windows/javadoc.html`.

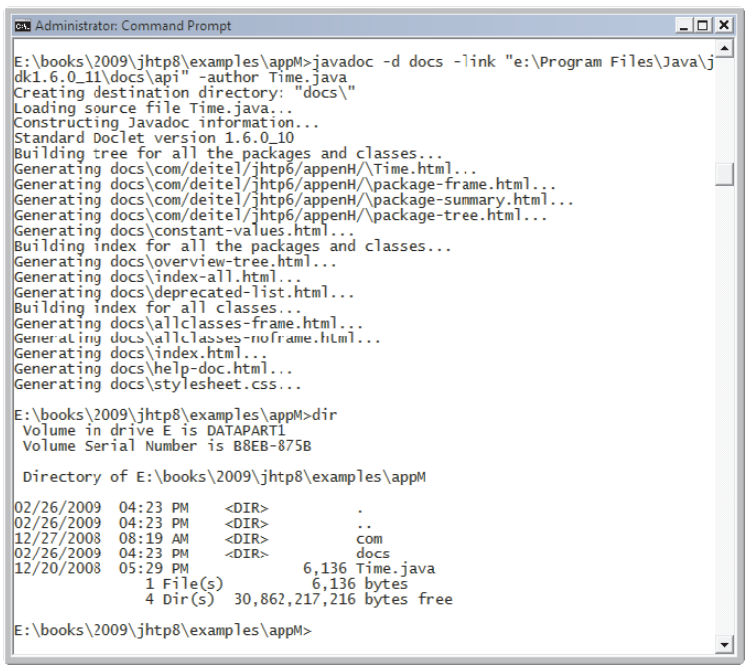


Figura M.7 | Utilizando a ferramenta `javadoc`.

Na Figura M.7, a opção **`-d`** especifica o diretório (por exemplo, `docs` dentro da pasta atual) no qual os arquivos HTML serão armazenados no disco. Usamos a opção **`-link`** para que nossa documentação tenha um link com a documentação da Sun (instalada no diretório `docs` dentro do diretório de instalação do JDK). Se a documentação da Sun estiver localizada em um diretório diferente, especifique esse diretório aqui; do contrário, você receberá um erro da ferramenta `javadoc`. Isso cria um hiperlink entre nossa documentação e a documentação da Sun (veja a Figura M.4, na qual a classe de Java `Exception` do pacote `java.lang` está hipervinculada). Sem o argumento `-link`, a `Exception` aparece como texto no documento HTML — não como um hyperlink à documentação da Java API para a classe `Exception`. A opção **`-author`** instrui o `javadoc` a processar o tag `@author` (ele ignora esse tag por padrão).

M.5 Arquivos produzidos por javadoc

Na última seção, executamos a ferramenta `javadoc` no arquivo `Time.java`. Quando `javadoc` executa, ele exibe o nome de cada arquivo de HTML que cria (veja a Figura M.7). A partir do arquivo-fonte, o `javadoc` criou um documento de HTML para a classe identificada `Time.html`. Se o arquivo-fonte contém múltiplas classes ou interfaces, um documento separado de HTML é criado para cada classe. Como a classe `Time` pertence a um pacote, a página será criada no diretório `docs\com\deitel\jhtp3\appenH` (em plataformas Windows). O diretório `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`**. Essa é a página HTML inicial da documentação. Para visualizar a documentação que você gera com o `javadoc`, carregue `index.html` em seu navegador da Web. Na Figura M.8, o frame direito contém a página `index.html` e o frame esquerdo contém a página **`allclasses-frame.html`** que, por sua vez, contém links para as classes do código-fonte. [*Nota*: Nosso exemplo não contém múltiplos pacotes, portanto não há nenhum frame listando os pacotes. Normalmente esse frame apareceria acima do frame esquerdo (contendo "All Classes"), como na Figura M.1.]

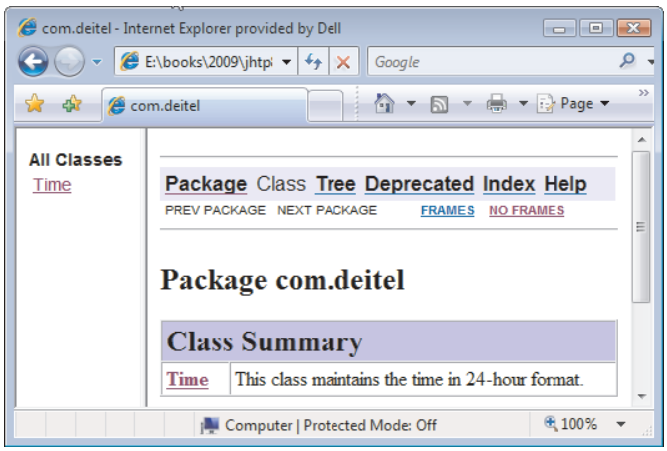
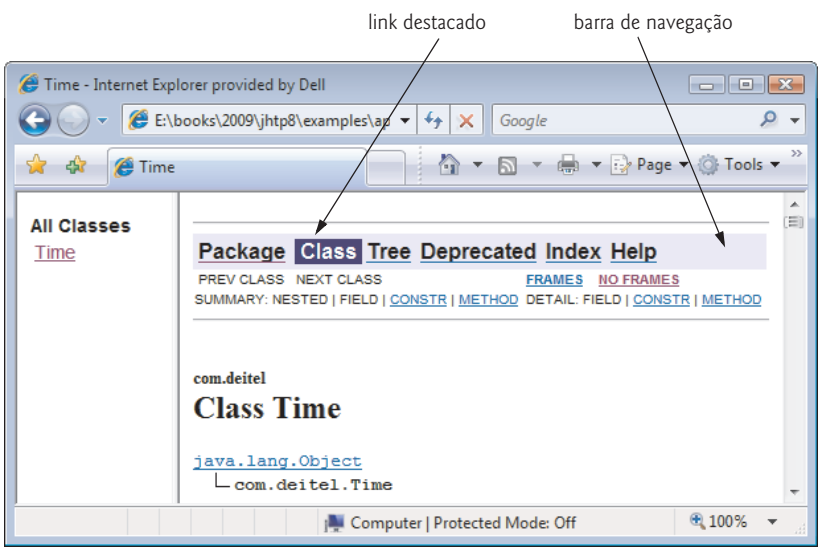


Figura M.8 | Página **Index**.

A Figura M.9 mostra a `index.html` da classe `Time`. Clique em **Time** no frame esquerdo para carregar a descrição da classe `Time`. A barra de navegação (na parte superior do frame direito) indica qual página HTML é atualmente carregada destacando o link da página (por exemplo, o link **Class**).



Figurar M.9 | Página **Class**.

Clicar no link **Tree** (Figura M.10) exibe uma hierarquia de classe para todas as classes exibidas no frame esquerdo. Em nosso exemplo, documentamos somente a classe `Time` — que estende `Object`. Clicar no link **Deprecated** vincula carrega **`deprecated-list.html`** no quadro direito. Essa página contém uma lista de todos os nomes obsoletos. Uma vez que não utilizamos o tag `@deprecated` nesse exemplo, essa página não contém nenhuma informação.

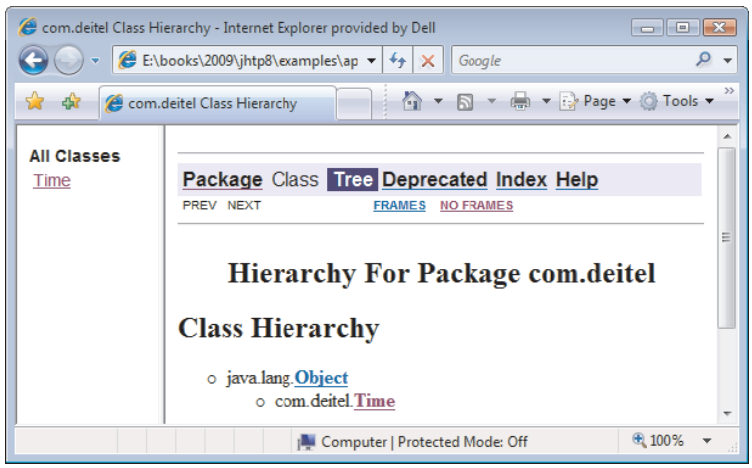


Figura M.10 | Página **Tree**.

Clicar no link **Index** carrega a página **`index-all.html`** (Figura M.11), que contém uma lista em ordem alfabética de todas as classes, interfaces, métodos e campos. Clicar no link **Help** carrega **`helpdoc.html`** (Figura M.12). Este é um arquivo de ajuda para navegar pela documentação. Um arquivo padrão de ajuda é fornecido, mas o programador pode especificar outros arquivos de ajuda.

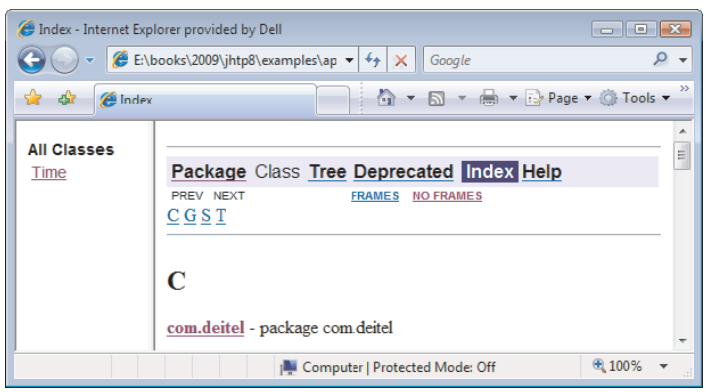


Figura M.11 | Página **Index**.

Entre os outros arquivos gerados por `javadoc` estão **`serialized-form.html`**, que documenta as classes `Serializable` e `Externalizable`, e **`package-list`**, um arquivo de texto, em vez de um arquivo HTML, que lista os nomes de pacotes e, na verdade, não faz parte da documentação. O arquivo `package-list` é utilizado pelo argumento de linha de comando `-link` para resolver referências cruzadas externas, isto é, permitir a vinculação de outras documentações a essa documentação.

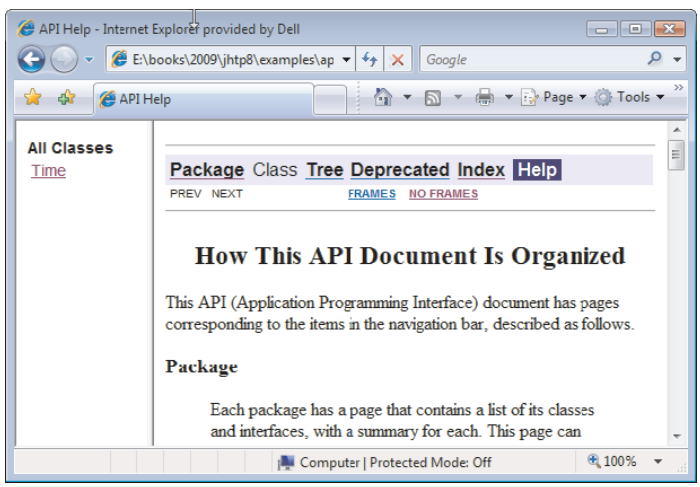


Figura M.12 | Página **Help**.