APÉNDICE

A

Trabajar con un proyecto BlueJ

A.1 Instalar BlueJ

Para trabajar con BlueJ se debe instalar el kit de desarrollo de Java 2 Standard Edition (J2SE SDK) y el entorno BlueJ.

Se puede encontrar el software J2SE SDK y las instrucciones detalladas para su instalación en el CD que acompaña este libro o bien en

```
http://java.sun.com/j2se/
```

Se puede encontrar el entorno BlueJ y las instrucciones para su instalación en el CD que acompaña este libro o bien en

```
http://www.bluej.org/
```

A.2 Abrir un proyecto

Para usar cualquiera de los proyectos de ejemplo incluidos en el CD que acompaña a este libro, se deben copiar previamente a un disco en el que se pueda grabar (por ejemplo, al disco duro). Los proyectos BlueJ se pueden abrir directamente desde el CD pero no se pueden ejecutar desde él. Cuando BlueJ ejecuta un proyecto, necesita grabar información en la carpeta que lo contiene y este es el motivo por el que generalmente, no resulta adecuado utilizar los proyectos directamente desde el CD.

La manera más fácil de usar los proyectos es copiar al disco duro la carpeta que contiene todos los proyectos del libro (de nombre *projects*).

Después de instalar e iniciar BlueJ haciendo doble clic sobre su icono, se selecciona la opción *Open...* del menú *Project*, se navega hasta la carpeta *projects* y se selecciona un proyecto. Se pueden abrir varios proyectos simultáneamente.

Se incluye más información sobre el uso de BlueJ en el Tutorial de BlueJ[1] que está en el CD del libro, al que también se puede acceder mediante la opción *BlueJ Tutorial* del menú *Help* de BlueJ.

A.3 El depurador de BlueJ

Se puede encontrar información sobre el uso del depurador de BlueJ en el Apéndice G y en el Tutorial de BlueJ. El tutorial está incluido en el CD del libro y también se puede acceder a él mediante la opción *BlueJ Tutorial* del menú *Help* de BlueJ.

---

[1] N. del T. El Tutorial de BlueJ que se incluye en el CD está en idioma inglés. Si necesita una versión en español, puede encontrarla en el sitio http://www.bluej.org/doc/tutorial.html

A.4 Contenido del CD

En el CD que se incluye en este libro se encuentran los siguientes archivos y directorios:

Carpeta	Comentario
acrobat/	*Acrobat Reader para varios sistemas operativos. Acrobat Reader es un programa que muestra e imprime archivos en formato PDF. Se necesita para leer o imprimir el Tutorial de Bluej. (Puede ocurrir que Acrobat Reader ya esté instalado; sólo se debe instalar si no se puede abrir el tutorial.)*
mac/	*Acrobat Reader para el S.O. Mac X.*
linux/	*Acrobat Reader para el S.O. Linux.*
solaris/	*Acrobat Reader para el S.O. Solaris.*
windows/	*Acrobat Reader para Microsoft Windows (todas las versiones).*
bluej/	*El sistema BlueJ y su documentación.*
bluejsetup-212.exe	*Instalador de BlueJ para Microsoft Windows (todas las versiones).*
bluej-212.zip	*BlueJ para S.O. Mac X.*
bluej-212.jar	*BlueJ para otros sistemas operativos.*
tutorial.pdf	*Tutorial de BlueJ.*
index.html	*Documentación del CD. Para leer este archivo, se debe abrir mediante un navegador. Contiene una visión global del CD, instrucciones de instalación y otras cuestiones útiles.*
j2sdk/	*Contiene el sistema Java 2 (Java 2 SDK) para varios sistemas operativos.*
linux/	*Instalador de Java 2 SDK para Linux.*
solaris/	*Instalador de Java 2 SDK para Solaris.*
windows/	*Instalador de Java 2 SDK para Microsoft Windows (todas las versiones.)*
j2sdk-doc/	*Contiene la documentación de la biblioteca de Java 2. Es un archivo de tipo zip. Para usar la documentación, se puede copiar este archivo al disco rígido y descomprimirlo.*
projects/	*Contiene todos los proyectos que se utilizan en este libro. Antes de usar los proyectos, se debe copiar esta carpeta completa al disco rígido. Contiene una subcarpeta para cada capítulo.*
runthis.exe	*Programa que utiliza la característica auto-abrir de Microsoft Windows (no es relevante para este libro).*
intro/	*Archivos de soporte para la documentación del CD. No es necesario usar directamente los archivos de esta carpeta, en su lugar se puede usar el archivo* index.html.

APÉNDICE

B Tipos de dato en Java

Java reconoce dos categorías de tipos: tipos primitivos y tipos objeto. Los tipos primitivos se almacenan directamente en las variables y tienen valores semánticos (se copian los valores cuando se asignan a otra variable). Los tipos objeto se almacenan mediante referencias al objeto (no se almacena el objeto propiamente dicho); cuando se asignan a otra variable sólo se copia la referencia, no el objeto.

B.1 Tipos primitivos

En la siguiente tabla se listan todos los tipos primitivos del lenguaje Java:

Nombre del tipo	Descripción	Ejemplos de literales
Números enteros		
`byte`	entero de 1 byte de tamaño (8 bit)	`24` `-2`
`short`	entero corto (16 bit)	`137` `-119`
`int`	entero (32 bit)	`5409` `-2003`
`long`	entero largo (64 bit)	`423266353L` `55L`
Números reales		
`float`	punto flotante de simple precisión	`43.889F`
`double`	punto flotante de doble precisión	`45.63` `2.4e5`
Otros tipos		
`char`	un solo carácter (16 bit)	`'m'` `'?'` `'\u00F6'`
`boolean`	un valor lógico (verdadero o falso)	`true` `false`

Notas:

- Un número que no contiene un punto decimal se interpreta generalmente como un `int`, pero se convierte automáticamente a los tipos `short`, `byte` o `long` cuando se le asigna (si el valor encaja). Se puede declarar un literal como `long` añadiendo una 'L' al final del número (también se puede utilizar la letra 'l' (L minúscula) pero debería evitarse ya que se puede confundir fácilmente con el uno).
- Un número con un punto decimal se considera de tipo `double`. Se puede especificar un literal como un `float` añadiendo una 'F' o 'f' al final del número.
- Un carácter se puede escribir como un carácter Unicode encerrándolo entre comillas simples o como un valor Unicode de cuatro dígitos precedidos por '`\u`'.
- Los dos literales booleanos son `true` y `false`.

Debido a que las variables de tipos primitivos no hacen referencia a objetos, no existen métodos asociados con los tipos primitivos. Sin embargo, cuando se usa un tipo primitivo en un contexto que requiere un tipo objeto se puede usar el proceso de *autoboxing* para convertir un valor primitivo en su correspondiente objeto. Para más detalles, recurra a la Sección B.3.

La siguiente tabla detalla los valores mínimo y máximo disponibles para los tipos numéricos.

Tipo	Mínimo	Máximo
byte	-128	127
short	-32768	32767
int	-2147483648	2147483647
long	-9223372036854775808	9223372036854775807
	Mínimo positivo	**Máximo positivo**
float	1.4e-45	3.4028235e38
double	4.9e-324	1.7976931348623157e308

B.2 Tipos objeto

Todos los tipos que no aparecen en la sección *Tipos primitivos* son tipos objeto. Esto incluye los tipos clase e interface de la biblioteca estándar de Java (como por ejemplo, `String`) y los tipos definidos por el usuario.

Una variable de tipo objeto contiene una referencia (o un «puntero») a un objeto. Las asignaciones y los pasajes de parámetros utilizan referencias semánticas (es decir, se copia la referencia, no el objeto). Después de asignar una variable a otra, ambas variables hacen referencia al mismo objeto. Se dice que las dos variables son alias del mismo objeto.

Las clases son las plantillas de los objetos: definen los campos y los métodos que poseerá cada instancia.

Los arreglos (*arrays*) se comportan como tipos objeto; también utilizan referencias semánticas.

B.3 Clases «envoltorio»

En Java, cada tipo primitivo tiene su correspondiente clase «envoltorio» que representa el mismo tipo pero que en realidad, es un tipo objeto. Estas clases hacen posible que se usen valores de tipos primitivos en los lugares en que se requieren tipos objeto mediante un proceso conocido como *autoboxing*. La siguiente tabla enumera los tipos primitivos y sus correspondientes clases envoltorio del paquete `java.lang`. Excepto `Integer` y `Character`, los nombres de las clases envoltorio coinciden con los nombres de los tipos primitivos, pero con su primera letra en mayúscula.

Tipo primitivo	Tipo envoltorio
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Siempre que se use un valor de un tipo primitivo en un contexto que requiera un tipo objeto, el compilador utiliza la propiedad de *autoboxing* para encapsular automáticamente al valor de tipo primitivo en un objeto envoltorio equivalente. Esto quiere decir, por ejemplo, que los valores de tipos primitivos se pueden agregar directamente en una colección. La operación inversa (*autounboxing*) también se lleva a cabo automáticamente cuando se utiliza un objeto envoltorio en un contexto que requiere un valor del tipo primitivo correspondiente.

APÉNDICE

C

Estructuras de control en Java

C.1 Sentencias de selección

If-else

La sentencia *if-else* tiene dos formas:

```
if (expresión) {
     sentencias
}
```

```
if (expresión) {
     sentencias
}
else {
     sentencias
}
```

Ejemplos:

```
if (campo.size() == 0) {
   System.out.println("El campo está vacío");
}

if (numero < 0) {
   informarError();
}
else {
   procesarNumero(numero);
}

if (numero < 0) {
   procesarNegativo();
}
else if (numero == 0) {
   procesarCero();
}
else {
   procesarPositivo();
}
```

switch

La sentencia *switch* selecciona un único valor de un número arbitrario de casos. Existen dos esquemas posibles:

```
switch (expresión) {
    case valor: sentencias;
                break;
    case valor: sentencias;
                break;
    (se omiten los restantes
casos)
    default: sentencias;
                break;
}
```

```
switch (expresión) {
    case valor1:
    case valor2:
    case valor3:
        sentencias;
        break;
    case valor4:
    case valor5:
        sentencias;
        break;
    (se omiten los restantes
casos)
    default:
        sentencias;
        break;
}
```

Notas:

- Una sentencia *switch* puede tener cualquier número de etiquetas `case`.
- La instrucción `break` después de cada `case` es necesaria; en caso contrario la ejecución continúa pasando a través de las sentencias de la etiqueta siguiente. La segunda forma descrita anteriormente usa este esquema. En este caso, los tres primeros valores ejecutarán la primera sección de sentencias mientras que los valores cuatro y cinco ejecutarán la segunda sección de sentencias.
- El caso `default` es opcional. Si no se da ningún valor por defecto puede ocurrir que este caso no se ejecute nunca.
- No es necesaria la instrucción `break` al final del caso por `default` (o del último `case`, si es que no hay sección `default`) pero se considera de buen estilo incluirla.

Ejemplos:

```
switch (dia) {
  case 1:   stringDia = "Lunes";
            break;
  case 2:   stringDia = "Martes";
            break;
  case 3:   stringDia = "Miércoles";
            break;
  case 4:   stringDia = "Jueves";
            break;
  case 5:   stringDia = "Viernes";
            break;
  case 6:   stringDia = "Sábado";
            break;
  case 7:   stringDia = "Domingo";
            break;
  default: stringDia = "Día no válido";
            break;
}

switch (mes) {
  case 1:
  case 3:
  case 5:
  case 7:
  case 8:
```

```
    case 10:
    case 12:
      numeroDeDias = 31;
      break;
    case 4:
    case 6:
    case 9:
    case 11:
      numeroDeDias = 30;
      break;
    case 2:
      if (esAnioBisiesto())
        numeroDeDias = 29;
      else
        numeroDeDias = 28;
      break;
}
```

C.2 Ciclos

Java tiene tres tipos de ciclos: *while*, *do-while* y *for*.

while

El *ciclo while* ejecuta un bloque de sentencias tantas veces como la evaluación de la expresión resulte verdadera. La expresión se evalúa antes de la ejecución del cuerpo del ciclo, por lo tanto, el cuerpo del ciclo podría ejecutarse cero veces (es decir, no ejecutarse).

```
while (expresión) {
  sentencias
}
```

```
Ejemplos:
  int i = 0;
  while (i < texto.size()) {
    System.out.println(texto.get(i));
    i++;
  }

  while (iter.hasNext()) {
    procesarObjeto(iter.next());
  }
```

do-while

El *ciclo do-while* ejecuta un bloque de sentencias tantas veces como la expresión resulte verdadera. La expresión es evaluada después de la ejecución del cuerpo del ciclo, por lo que el cuerpo de este ciclo se ejecuta siempre por lo menos una vez.

```
do {
      sentencias
} while (expresión);
```

```
Ejemplo:
```

```
do {
   entrada = leerEntrada();
   if (entrada == null) {
      System.out.println("Pruebe nuevamente");
   }
} while (entrada == null);
```

for

El *ciclo for* tiene dos formas diferentes. La primera se conoce también como *ciclo for-each* y se usa exclusivamente para recorrer los elementos de una colección. A la variable del ciclo se le asigna el valor de los sucesivos elementos de la colección en cada iteración del ciclo.

```
for (declaración-de-variable : colección) {
     sentencias
}
Ejemplo:
for (String nota : lista) {
      System.out.println(nota);
}
```

La segunda forma del *ciclo for* ejecuta un bloque de sentencias tantas veces como la condición se evalúe verdadera. Antes de iniciar el ciclo, se ejecuta exactamente una vez, una sentencia de *inicialización*. La condición es evaluada antes de cada ejecución del cuerpo del ciclo (por lo que el cuerpo del ciclo podría no ejecutarse). Se ejecuta una sentencia de *incremento* al finalizar cada ejecución del cuerpo del ciclo.

```
for (inicialización; condición; incremento) {
      sentencias
}
```

Ejemplo:

```
for(int i = 0; i < texto.size(); i++) {
     System.out.println(texto.get(i));
   }
```

C.3 Excepciones

El lanzamiento y la captura de excepciones proporciona otro par de construcciones que alteran el flujo del control.

```
try {
   sentencias
}
catch (tipo-de-excepción nombre) {
   sentencias
}
finally {
   sentencias
}
```

Ejemplo:

```
try {
```

```
    FileWriter writer = new FilwWriter("foo.txt");
    writer.write(texto);
    writer.close();
}
catch (IOException e) {
    Debug.reportError("Falló la grabación del texto");
    Debug.reportError("La excepción es: " + e );
}
```

Una sentencia de excepción puede tener cualquier número de cláusulas *catch* que son evaluadas en el orden en que aparecen y se ejecuta sólo la primera cláusula que coincida. (Una cláusula coincide si el tipo dinámico del objeto excepción que ha sido lanzado es compatible en la asignación con el tipo de excepción declarado en la cláusula *catch*.) La cláusula *finally* es opcional.

C.4 Aserciones

Hay dos formas de sentencias de aserción:

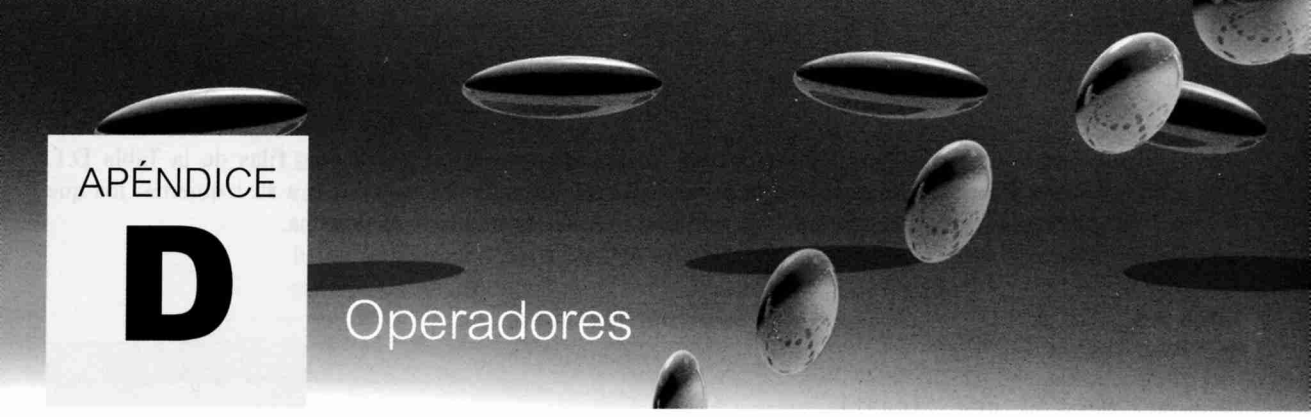
```
assert expresión-booleana;
assert expresión-booleana : expresión;
```

Ejemplos:

```
assert getDatos(clave) != null;

assert esperado = actual :
    " El valor actual: " + actual +
          " no coincide con el valor esperado: " + esperado;
```

Si la expresión de la aserción se evalúa falsa, se disparará un `AssertionError`.



APÉNDICE

D Operadores

D.1 Expresiones aritméticas

Java dispone de una cantidad considerable de operadores para expresiones aritméticas y lógicas. La tabla D.1 muestra todo aquello que se clasifica como un operador, incluyendo la conversión de tipos (*casting*) y el pasaje de parámetros. Los principales operadores aritméticos son:

+	*suma*
-	*resta*
*	*multiplicación*
/	*división*
%	*módulo o resto de una división entera*

Tanto en la división como en el módulo, los resultados de las operaciones dependen de si sus operandos son enteros o si son valores de punto flotante. Entre dos valores enteros, la división retiene el resultado entero y descarta cualquier resto; pero entre dos valores de punto flotante, el resultado es un valor de punto flotante:

`5 / 3` da por resultado `1`
`5.0 / 3` da por resultado `1.6666666666666667`

(Observe que es necesario que uno sólo de los operandos sea de punto flotante para que se produzca un resultado de punto flotante.)

Cuando en una operación aparecen más operadores, se deben usar las *reglas de precedencia* para indicar el orden de su aplicación. En la Tabla D.1, los operadores se presentan por nivel de precedencia, de mayor a menor (en la primera fila aparecen los operadores de nivel de precedencia más alto). Por ejemplo, podemos ver que la multiplicación, la división y el módulo preceden a la suma y a la resta y esta es la razón por la que los dos ejemplos siguientes dan por resultado 100:

```
51 * 3 - 53
154 - 2 * 27
```

Los operadores que tienen el mismo nivel de precedencia se evalúan de izquierda a derecha.

Se pueden usar paréntesis cuando se necesite alterar el orden de evaluación. Es por este motivo que los dos ejemplos siguientes dan por resultado 100:

```
(205 - 5) / 2
2 * (47 + 3)
```

Observe que algunos operadores aparecen en las dos primeras filas de la Tabla D.1. Los que aparecen en la primera fila admiten un solo operando a su izquierda; los que están en la segunda fila admiten un solo operando a su derecha.

Tabla D.1 Operadores Java por nivel de precedencia (de mayor a menor)

```
[]    .     ++    - -  (parámetros)
++    - -   +     -     !     "
new   (cast)
*     /     %
+     -
<<    >>    >>>
<     >     >=    <=    instanceof
==    !=
&
^
|
&&
||
?:
=     +=    - =   *=    /=    %=    >>=   >>>=  &=          |=     ^=
```

D.2. Expresiones lógicas

En las expresiones lógicas, se usan los operadores para combinar operandos y producir un único valor lógico, ya sea verdadero o falso (*true* o *false*). Las expresiones lógicas generalmente se encuentran en las condiciones de las sentencias *if-else* y en las de los ciclos.

Los operadores relacionales o de comparación combinan generalmente un par de operandos aritméticos, aunque también se utilizan para evaluar la igualdad y la desigualdad de referencias a objetos. Los operadores relacionales de Java son:

==	*igual*	!=	*distinto*
<	*menor*	<=	*menor o igual*
>	*mayor*	>=	*mayor o igual*

Los operadores lógicos binarios combinan dos expresiones lógicas para producir otro valor lógico. Los operadores son:

&& *y (and)*
|| *o (or)*
^ *o excluyente*

Y además,

! *no (not)*

que toma una expresión lógica y cambia su valor de verdadero a falso y viceversa.

La manera en que se aplican los operadores && y || es un poco extraña. Si el operando izquierdo es falso entonces resulta irrelevante el valor del operando derecho y no será evaluado; de igual manera, si el operando izquierdo es verdadero, no será evaluado el operando derecho. Por este motivo, se conoce a estos operadores como operadores en «cortocircuito».

APÉNDICE

E

Ejecutar Java fuera del entorno BlueJ

A lo largo de este libro hemos usado BlueJ para desarrollar y ejecutar nuestras aplicaciones Java. Hay una buena razón para esto: BlueJ nos ofrece algunas herramientas para que resulten más fáciles algunas de las tareas de desarrollo. En particular, nos permite ejecutar fácilmente métodos individuales de clases y de objetos, lo que resulta muy útil si queremos probar rápidamente un fragmento de código.

Dividimos la discusión sobre cómo trabajar fuera del entorno BlueJ en dos categorías: ejecutar una aplicación y desarrollarla fuera del entorno BlueJ.

E.1 Ejecutar fuera del entorno BlueJ

Generalmente, cuando se entregan las aplicaciones a los usuarios finales, son ejecutadas de diferentes maneras. Las aplicaciones tienen un solo punto de comienzo que define el lugar en que empieza la ejecución cuando un usuario inicia la aplicación.

El mecanismo exacto que se usa para iniciar una aplicación depende del sistema operativo; generalmente, se hace doble clic sobre el icono de la aplicación o se ingresa el nombre de la misma en una línea de comando. Luego, el sistema operativo necesita saber qué método o qué clase debe invocar para ejecutar el programa completo.

En Java, este problema se resuelve usando una convención: cuando se inicia un programa Java, el nombre de la clase se especifica como un parámetro del comando de inicio y el nombre del método es siempre el mismo, el nombre de este método es «main». Por ejemplo, considere el siguiente comando ingresado en una línea de comando, como si fuera un comando de Windows o de una terminal Unix:

```
java Juego
```

El comando `java` inicia la máquina virtual de Java, que forma parte del kit de desarrollo de Java (SDK) y que debe estar instalado en su sistema. `Juego` es el nombre de la clase que queremos iniciar.

Luego, el sistema Java buscará un método en la clase `Juego` cuya signatura coincida exactamente con la siguiente:

```
public static void main(String[] args)
```

El método debe ser público para que pueda ser invocado desde el exterior de la clase. Debe ser estático porque no existe ningún objeto cuando se inicia el programa; inicialmente, tenemos sólo clases, motivo por el cual sólo podemos invocar métodos estáticos. Este método estático crea el primer objeto. El tipo de retorno es `void` ya que este método no retorna ningún valor. Aunque el nombre «`main`» fue seleccionado arbitrariamente por los desarrolladores de Java, es fijo: el método debe tener siempre este

nombre. (La elección de «main» como nombre del método inicial en realidad proviene del lenguaje C, del que Java hereda gran parte de su sintaxis.)

El parámetro es un arreglo de String, que permite a los usuarios pasar argumentos adicionales. En nuestro ejemplo, el valor del parámetro args será un arreglo de longitud cero. Sin embargo, la línea de comandos que inicia el programa puede definir argumentos:

```
java Juego 2 Fred
```

En esta línea de comando, cada palabra ubicada a continuación del nombre de la clase será leído como un String independiente y pasado al método main como un elemento del arreglo de String. En este caso, el arreglo args contendrá dos elementos que son las cadenas «2» y «Fred». Los parámetros en la línea de comandos no son muy usados en Java.

En teoría, el cuerpo del método main puede contener el número de sentencias que se deseen. Sin embargo, un buen estilo indica que el método main debiera mantenerse lo más corto posible; específicamente, no debiera contener nada que forme parte de la lógica de la aplicación.

En general, el método main debe hacer exactamente lo que se hizo interactivamente para iniciar la misma aplicación en BlueJ. Por ejemplo, si para iniciar la aplicación en BlueJ se creó un objeto de la clase Juego y se invocó el método de nombre start, en el método main de la clase Juego deberían agregarse las siguientes sentencias:

```
public static void main (String[] args)
{
  Juego juego = new Juego();
  juego.start();
}
```

Ahora, al ejecutar el método main se imitará la invocación interactiva del juego.

Los proyectos Java se guardan generalmente en un directorio independiente para cada uno y todas las clases del proyecto se ubican dentro de este directorio. Cuando se ejecute el comando para iniciar Java y ejecutar su aplicación, se debe asegurar de que el directorio del proyecto sea el directorio activo en la terminal de comandos, lo que asegura que se encontrarán las clases que se usan.

Si no puede encontrar una clase específica, la máquina virtual de Java generará un mensaje de error similar a este:

```
Exception in thread "main" java.lang.NoClassDefFoundError: Juego
```

Si ve un mensaje como éste, asegúrese de que escribió correctamente el nombre de la clase y de que el directorio actual realmente contenga esta clase. La clase se guarda en un archivo de extensión ".class": por ejemplo, el código de la clase Juego se almacena en un archivo de nombre Juego.class.

Si encuentra la clase pero ésta no contiene un método main (o el método main no posee la signatura correcta) verá un mensaje similar a este:

```
Exception in thread "main" java.lang.NoSuchMethodError: main
```

En este caso, asegúrese de que la clase que quiere ejecutar tenga el método main correcto.

E.2 Crear archivos ejecutables .jar

Los proyectos Java se almacenan como una colección de archivos en un directorio (o carpeta). A continuación, hablaremos brevemente sobre los diferentes tipos de archivo.

Generalmente, para distribuir aplicaciones a otros usuarios es más fácil si toda la aplicación se guarda en un único archivo; el mecanismo de Java que realiza esto tiene el formato de archivo Java («.jar»). Todos los archivos de una aplicación se pueden reunir en un único archivo y aun así podrán ser ejecutados. (Si está familiarizado con el formato de compresión «zip», sería interesante saber que, de hecho, el formato es el mismo. Los archivos jar pueden abrirse mediante programas zip y viceversa.)

Para crear un archivo .jar ejecutable es necesario especificar la clase principal en algún lugar. (Recuerde: el método que se ejecuta siempre es el `main`, pero necesitamos especificar la clase que lo contiene.) Esta especificación se hace incluyendo un archivo de texto en el archivo .jar (el archivo explícito) con la información necesaria. Afortunadamente, BlueJ se ocupa por su propia cuenta de esta tarea.

Para crear un archivo ejecutable .jar en BlueJ use la función *Project – Export* y especifique la clase que contiene el método `main` en la caja de diálogo que aparece. (Debe escribir un método `main` exactamente igual al descrito anteriormente.)

Para ver detalles sobre esta función, lea el Tutorial de BlueJ al que puede acceder mediante el menú *Help-Tutorial* de BlueJ o bien visitando el sitio web de BlueJ.

Una vez que se creó el archivo ejecutable .jar, se puede ejecutar haciendo doble clic sobre él. La computadora que ejecuta este archivo .jar debe tener instalado el JDK (Java Development Kit) o el JRE (Java Runtime Environment) y asociado con archivos .jar.

E.3 Desarrollar fuera del entorno BlueJ

Si no quiere solamente ejecutar programas, sino que también quiere desarrollarlos fuera del entorno BlueJ, necesitará editar y compilar las clases. El código de una clase se almacena en un archivo de extensión «`.java`»; por ejemplo, la clase `Juego` se almacena en un archivo de nombre `Juego.java`. Los archivos fuente pueden editarse con cualquier editor de textos. Existen muchos editores de textos libres o muy baratos. Algunos, como el *Notepad* o el *WordPad* se distribuyen con Windows, pero si en realidad quiere usar un editor para hacer algo más que una prueba rápida, querrá obtener uno mejor. Sin embargo, sea cuidadoso con los procesadores de texto: generalmente los procesadores de texto no graban en formato de texto plano y Java no podrá leerlos.

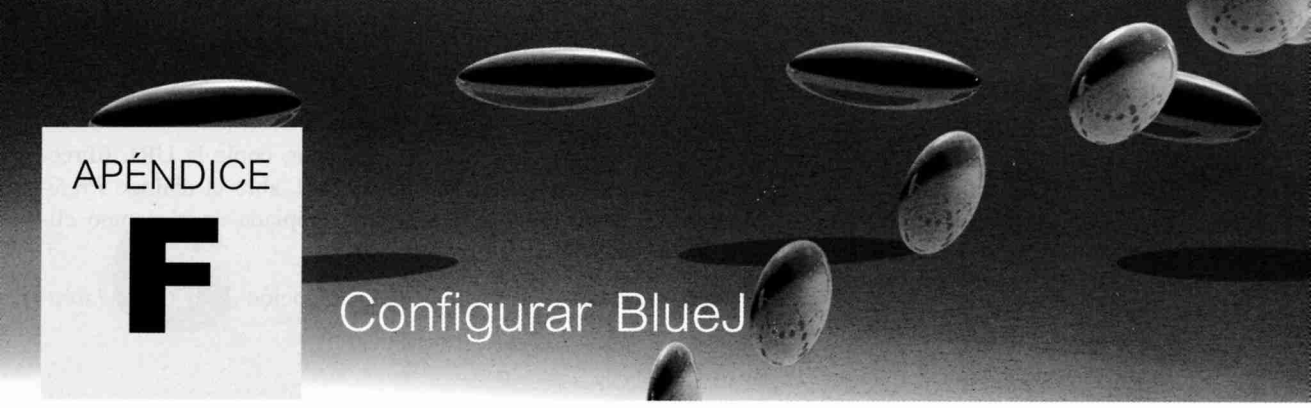
Los archivos fuente pueden compilarse desde una línea de comando usando el compilador Java que se incluye en el JDK y que se invoca mediante el comando `javac`. Para compilar un archivo fuente de nombre `Juego.java` use el comando

```
javac Juego.java
```

Este comando compilará la clase `Juego` y cualquier otra clase que dependa de ella; creará un archivo denominado `Juego.class` que contiene el código que puede ser ejecutado mediante la máquina virtual de Java. Para ejecutar este archivo use el comando

```
java Juego
```

Observe que este comando no incluye la extensión del archivo «.class».



APÉNDICE

F Configurar BlueJ

Se pueden configurar muchos de los valores de BlueJ para que se adapte mejor a su situación personal. Algunas opciones de configuración están disponibles mediante la caja de diálogo *Preferences* del sistema BlueJ, pero es posible acceder a muchas otras opciones editando el «archivo de definiciones de BlueJ» que está ubicado en *<bluej_home>/lib/bluej.defs*, donde *<bluej_home>* es la carpeta donde se encuentra instalado BlueJ.

Los detalles de configuración se explican en la sección «*Tips archive*» del sitio web de BlueJ a la que se puede acceder en la dirección

```
http://www.bluej.org/help/archive.html
```

A continuación presentamos las cosas más comunes que la gente suele cambiar. Puede encontrar muchas más opciones de configuración leyendo el archivo *bluej.defs*.

F.1 Cambiar el idioma de la interfaz

Puede cambiar el idioma de la interfaz por cualquiera de los idiomas disponibles. Para hacerlo, abra el archivo *bluej.defs* y busque la línea que dice

```
bluej.language=english
```

y cámbiela por uno de los idiomas disponibles. Por ejemplo:

```
bluej.language=spanish
```

Los comentarios en el archivo de definición enumeran todos los lenguajes disponibles. Como mínimo se incluyen los idiomas afrikáans, chino, checo, inglés, francés, alemán, italiano, japonés, coreano, portugués, español y sueco.

F.2 Usar la documentación API en forma local

Puede usar una copia local de la documentación de la biblioteca de clases de Java (API). De esta manera, el acceso a la documentación es más rápido y puede usar la documentación sin tener que estar conectado a Internet.

Para hacerlo, copie el archivo de documentación de Java desde el CD (un archivo zip) y descomprímalo en el lugar en que quiera guardar la documentación de Java; se creará una carpeta de nombre *docs*.

Luego abra un navegador y usando la función «*Abrir archivo...*» (o cualquier otra equivalente), abra el archivo *api/index.html* situado en la carpeta *docs*.

Una vez que se visualiza correctamente el API en el navegador, copie la URL (dirección web) del campo de dirección de su navegador, abra BlueJ, abra el diálogo *Preferences*, seleccione la ficha *Miscellaneous* y pegue la URL copiada en el campo etiquetado como *JDK documentation URL*.

Ahora podrá abrir una copia local del API seleccionando la opción *Java Class Libraries* del menú *Help*.

F.3 Cambiar las plantillas para las clases nuevas

Cuando crea una clase nueva, el código se presenta con un texto predeterminado que proviene de una plantilla. Se puede cambiar este texto de modo que se adapte a sus preferencias.

Las plantillas se almacenan en las carpetas

<bluej_home>/lib/<language>/templates/ y en
<bluej_home>/lib/<language>/templates/newclass/

en donde *<bluej_home>* es la carpeta de instalación de BlueJ y *<language>* es el idioma actualmente en uso (por ejemplo, *english*).

Las plantillas son archivos de texto y se pueden editar mediante cualquier editor de textos estándar.

APÉNDICE

G

Usar el depurador

El depurador de BlueJ proporciona un conjunto de funcionalidades básicas de depuración intencionalmente simplificadas y que son genuinamente útiles tanto para la depuración de programas como para alcanzar mayor comprensión del comportamiento de la ejecución de un programa.

Se puede acceder a la ventana del depurador seleccionando el elemento *Show Debugger* del menú *View* o presionando el botón derecho del *ratón* sobre el indicador de trabajo y seleccionando *Show Debugger* desde el menú contextual. La Figura G.1 muestra la ventana del depurador.

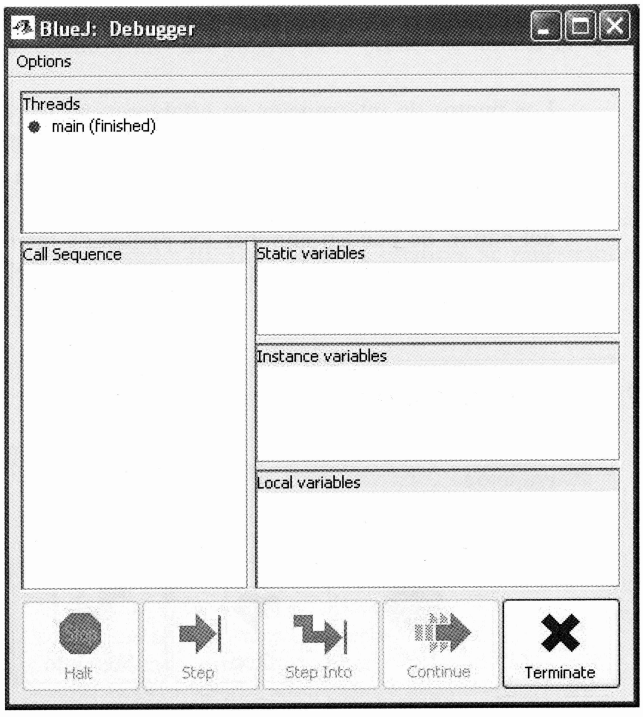


Figura G.1
La ventana del depurador de BlueJ

La ventana del depurador tiene cinco zonas de visualización y cinco botones de control. Las zonas de visualización y los botones se activan solamente cuando un programa alcanza un punto de interrupción o se para por alguna otra razón. Las siguientes secciones describen cómo establecer puntos de interrupción para controlar la ejecución de un programa y el propósito de cada una de las zonas.

G.1 Puntos de interrupción

Un punto de interrupción es un bandera que se asocia con una línea de código (Figura G.2). Cuando se alcanza un punto de interrupción durante la ejecución de un programa, se activan las zonas de visualización y los controles del depurador permitiendo inspeccionar el estado del programa y controlar la ejecución a partir de allí.

Figura G.2
Un punto de interrupción asociado con una línea de código

```
32      /**
33       * Imprime el siguiente mensaje (si es que hay alguno) para este
34       * usuario en la terminal de texto.
35       */
36      public void imprimirMensajeSiguiente()
37      {
38          Mensaje unMensaje = servidor.getMensajeSiguiente(usuario);
39          if(unMensaje == null) {
40              System.out.println("No hay ningún mensaje nuevo.");
41          }
42          else {
43              unMensaje.imprimir();
44          }
45      }
```

Los puntos de interrupción se establecen en la ventana del editor, ya sea presionando el botón izquierdo del *ratón* en la zona de puntos de interrupción situada a la izquierda del código o bien ubicando el cursor en la línea de código en la que debiera estar el punto de interrupción y seleccionando la opción *Set/Clear Breakpoint* del menú *Tools* del editor. Se pueden eliminar los puntos de interrupción mediante el proceso inverso. Sólo se pueden fijar puntos de interrupción en el código de las clases que hayan sido previamente compiladas.

G.2 Los botones de control

La Figura G.3 muestra los botones de control que se activan ante un punto de interrupción.

Figura G.3
Botones de control activos ante un punto de interrupción

G.2.1 Halt

El botón *Halt* está activo cuando el programa se está ejecutando, para permitir que la ejecución se pueda interrumpir, de ser necesario. Si la ejecución se detiene, el depurador mostrará el estado del programa como si hubiera alcanzado un punto de interrupción.

G.2.2 Step

El botón *Step* ejecuta la sentencia actual. La ejecución se detendrá nuevamente cuando se complete dicha sentencia. Si la sentencia involucra una llamada a método, se completa la llamada al método antes de que la ejecución se detenga nuevamente (a menos que el método invocado tenga otro punto de interrupción explícito).

G.2.3 Step Into

El botón *Step Into* ejecuta la sentencia actual. Si esta sentencia es una llamada a un método entonces la ejecución se introducirá en ese método y se detendrá nuevamente en la primer sentencia del mismo.

G.2.4 Continue

El botón *Continue* continúa la ejecución del programa hasta que se alcance el siguiente punto de interrupción, se interrumpa la ejecución mediante el botón *Halt* o se complete la ejecución normalmente.

G.2.5 Terminate

El botón *Terminate* finaliza agresivamente la ejecución del programa actual de manera tal que no puede ser detenida nuevamente. Si se desea simplemente interrumpir la ejecución para examinar el estado actual del programa es preferible utilizar la operación *Halt*.

G.3 La zona de variables

La Figura G.4 muestra las tres zonas activas en las que se muestran las variables cuando se encuentra un punto de interrupción, en un ejemplo tomado de la simulación *predador-presa* trabajada en el Capítulo 10. Las variables estáticas se muestran en la zona superior, las variables de instancia en la del medio y las variables locales en la zona inferior.

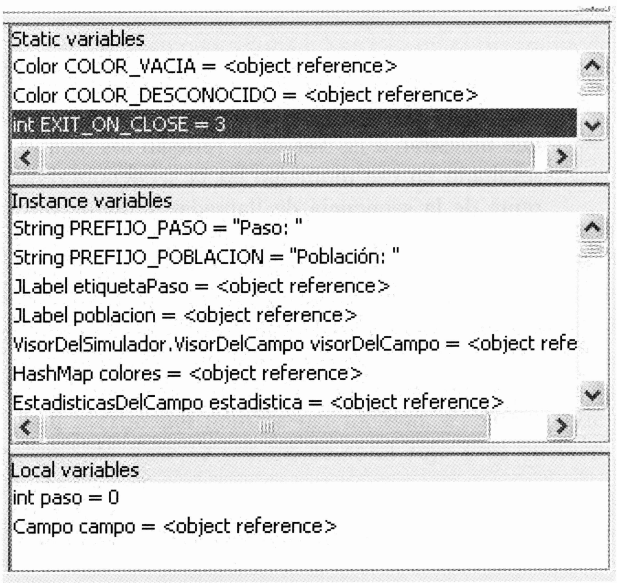


Figura G.4
Zonas de variables activas

Cuando se alcanza un punto de interrupción, la ejecución se detendrá en una sentencia de un objeto arbitrario dentro del programa actual. La zona de variables estáticas (*Static variables*) muestra los valores de las variables estáticas definidas en la clase de dicho objeto. La zona de variables de instancia (*Instance variables*) muestra las variables de instancia de dicho objeto en particular. Ambas zonas también incluyen las variables heredadas de las superclases.

La zona de *variables locales* (*Local variables*) muestra los valores de las variables locales y de los parámetros del método o del constructor que se está ejecutando actualmente. Las variables locales aparecerán en esta zona sólo una vez que hayan sido inicializadas ya que solamente comienzan a existir en la máquina virtual de Java a partir de ese momento.

G.4 La zona de Secuencia de llamadas

La Figura G.5 muestra la zona *Call Sequence* que contiene una secuencia de cuatro métodos de profundidad. Los métodos aparecen en la secuencia en el formato `Clase.método`, independientemente de si son métodos estáticos o métodos de instancia. Los constructores aparecen en la secuencia como `Clase.<init>`.

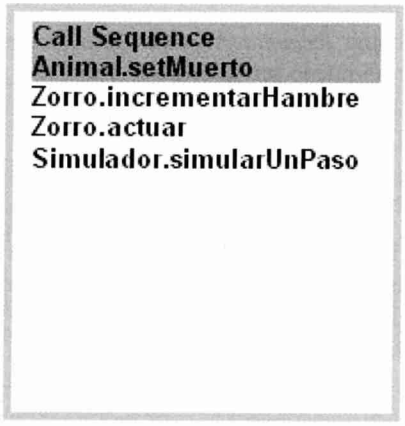


Figura G.5
Una secuencia de llamadas

La secuencia de llamadas opera como una pila: el método que aparece en la parte superior de la secuencia es donde reside actualmente el flujo de la ejecución. Las zonas que muestran a las variables reflejan los detalles del método o del constructor que esté resaltado en ese momento en la secuencia de llamadas. Al seleccionar una línea diferente de la secuencia de llamadas se actualizarán los contenidos de las otras zonas.

G.5 La zona de Threads

Esta zona está fuera del alcance de este libro y no será tratada.

APÉNDICE

H

Herramienta JUnit de pruebas unitarias

En este apéndice hacemos una breve revisión de las principales características que soporta BlueJ relacionadas con el estilo de pruebas unitarias creadas mediante JUnit. Se pueden encontrar más detalles sobre el tema en el tutorial que está disponible en el CD que acompaña este libro y en el sitio web de BlueJ.

H.1 Habilitar la funcionalidad de pruebas unitarias

Para habilitar la funcionalidad de pruebas unitarias de BlueJ es necesario asegurarse de que esté marcada la opción *Show unit testing tools* en el menú *Tools-Preferences-Miscellaneous*. Una vez marcada, la ventana principal de BlueJ contendrá algunos botones adicionales que se activan cuando se abre un proyecto.

H.2 Crear una clase de prueba

Se crea una clase de prueba haciendo clic con el botón derecho del *ratón* sobre una clase en el diagrama de clases y seleccionando la opción *Create Test Class*. El nombre de una clase de prueba se determina automáticamente agregando la palabra «Test» a modo de sufijo al nombre de la clase asociada. Alternativamente, puede crearse una clase de prueba seleccionando el botón *New Class...* y eligiendo *Unit Test* como el tipo de la clase. En este caso, la elección del nombre es totalmente libre.

Las clases de prueba se denotan con `<<unit test>>` en el diagrama de clases y tienen un color diferente del color de las clases ordinarias.

H.3 Crear un método de prueba

Los métodos de prueba se pueden crear interactivamente. Se puede grabar la secuencia de interacciones del usuario con el diagrama de clases y con el banco de objetos y luego capturarla como una secuencia de sentencias y de declaraciones Java en un método de la clase de prueba. Se comienza la grabación seleccionando la opción *Create Test Method* del menú contextual asociado con una clase de prueba. BlueJ solicitará el nombre del nuevo método. Si el nombre no comienza con la palabra `test` entonces se agregará como un prefijo del nombre del método. El símbolo de grabación a la izquierda del diagrama de clase se pondrá de color rojo y se vuelven disponibles los botones *End* y *Cancel*.

Una vez que comenzó la grabación, cualquier creación de objeto o llamadas a métodos formarán parte del código del método que se está creando. Seleccione *End* para com-

pletar la grabación y capturar la prueba o *Cancel* para descartar la grabación y dejar sin cambios a la clase de prueba.

H.4 Pruebas con aserciones

Mientras se graba un método de prueba, cualquier llamada a método que retorne un resultado abrirá una ventana del tipo *Method Result* que ofrece la oportunidad de evaluar el valor del resultado marcando la opción *Assert that*. El menú desplegable que aparece contiene un conjunto de aserciones posibles para el valor del resultado. Si se estableció una aserción, será codificada como una llamada a método en el método de prueba que dará un `AssertionError` en el caso en que la prueba falle.

H.5 Ejecutar pruebas

Los métodos se pueden ejecutar individualmente seleccionándolos del menú contextual asociado a la clase de prueba. Si una prueba resulta exitosa, se indicará mediante un mensaje en la línea de estado de la ventana principal. Si una prueba fracasa aparecerá la ventana *Test Results*. Al seleccionar *Test All* del menú contextual de la clase de prueba se ejecutarán todas las pruebas de una sola clase. En la ventana *Test Results* se detallará el éxito o el fracaso de cada método.

H.6 Conjunto de Objetos de prueba (fixtures)

Se puede capturar el contenido del banco de objetos como un «juego de prueba» (*fixture*), seleccionando la opción *Object Bench to Test Fixture* del menú contextual asociado con la clase de prueba. El efecto de crear un juego de prueba es que se agrega una definición de campo para cada objeto en la clase de prueba y se agregan las sentencias en su respectivo método `setup` que recreará el estado exacto de los objetos tal como estaban en el banco de objetos. Luego, los objetos son eliminados del banco.

El método `setUp` se ejecuta automáticamente antes de ejecutar cualquier método de prueba por lo que todos los objetos del juego de prueba estarán disponibles para todas las pruebas.

Los objetos del juego de prueba pueden ser creados nuevamente en el banco de objetos seleccionando la opción *Test Fixture to Object Bench* desde el menú de la clase de prueba.

APÉNDICE

I

El documentador de Java: javadoc

La escritura de buena documentación de las definiciones de las clases y de las interfaces es un complemento importante para obtener código de buena calidad. La documentación le permite al programador comunicar sus intenciones a los lectores humanos en un lenguaje natural de alto nivel, en lugar de forzarlos a leer código de nivel relativamente bajo. La documentación de los elementos públicos de una clase o de una interfaz tienen un valor especial, pues los programadores pueden usarla sin tener que conocer los detalles de su implementación.

En todos los proyectos de ejemplo de este libro hemos usado un estilo particular de comentarios que es reconocido por la herramienta de documentación `javadoc` que se distribuye como parte del kit de desarrollo (SDK) de Java de Sun Microsystem. Esta herramienta automatiza la generación de documentación de clases en formato HTML con un estilo consistente. El API de Java ha sido documentado usando esta misma herramienta y se aprecia su valor cuando se usa la biblioteca de clases.

En este apéndice hacemos un breve resumen de los principales elementos de los comentarios de documentación que deberá introducir habitualmente en su propio código fuente.

I.1 Comentarios de documentación

Los elementos de una clase que se documentarán son la definición de la clase, sus campos, constructores y métodos. Desde el punto de vista de un usuario, lo más importante de una clase es que tenga documentación sobre ella y sobre sus constructores y métodos públicos. Tendemos a no proporcionar comentarios del estilo de javadoc para los campos aunque recordamos que forman parte del detalle del nivel de implementación y no es algo que verán los usuarios.

Los comentarios de documentación comienzan siempre con los tres caracteres «/**» y terminan con el par de caracteres «*/». Entre estos símbolos, un comentario contendrá una **descripción principal** seguida por una sección de **etiqueta**, aunque ambas partes son opcionales.

I.1.1 La descripción principal

La descripción principal de una clase debiera consistir en una descripción del objetivo general de la clase. El Código I.1 muestra parte de una típica descripción principal, tomada de la clase `Juego` del proyecto *world-of-zuul*. Observe que la descripción incluye detalles sobre cómo usar esta clase para iniciar el juego.

Código I.1
La descripción principal de un comentario de clase

```
/**
 *  Esta es la clase principal de la aplicación "World of
Zuul"
 *  "World of Zuul" es un juego de aventuras muy sencillo,
basado en texto.
 *  Los usuarios pueden caminar por algún escenario, y eso
es todo lo
 *  que hace el juego. ¡Podría ampliarse para que resulte
más interesante!
 *  Para jugar, cree una instancia de esta clase e invoque
el método "jugar"
 */
```

La descripción principal de un método debiera ser bastante general, sin introducir demasiados detalles sobre su implementación. En realidad, la descripción principal de un método generalmente consiste en una sola oración, como por ejemplo

```
/**
*    Crea un nuevo pasajero con distintas ubicaciones de
salida y de destino.
*/
```

Las ideas esenciales debieran presentarse en la primera sentencia de la descripción principal de una clase, de una interfaz o de un método ya que es lo que se usa a modo de resumen independiente en la parte superior de la documentación generada.

Javadoc también soporta el uso de etiquetas HTML en sus comentarios.

I.1.2 La sección de etiquetas

A continuación de la descripción principal aparece la sección de etiquetas. Javadoc reconoce alrededor de 20 etiquetas pero sólo trataremos las más importantes (Tabla I.1). Las etiquetas pueden usarse de dos maneras: en bloques de etiquetas o como etiquetas de una sola línea. Sólo hablaremos de los bloques de etiquetas pues son los que se usan con mayor frecuencia. Para ver más detalles sobre las etiquetas de una sola línea y sobre las restantes etiquetas, puede recurrir a la sección *javadoc* de la documentación *Tools and Utilities* que forma parte del Java SDK.

Tabla I.1
Etiquetas más comunes de javadoc

Etiqueta	Texto asociado
`@author`	nombre(s) del autor(es)
`@param`	nombre de parámetro y descripción
`@return`	descripción del valor de retorno
`@see`	referencia cruzada
`@throws`	tipo de excepción que se lanza y las circunstancias en las que se hace
`@version`	descripción de la versión

Las etiquetas `@author` y `@version` se encuentran regularmente en los comentarios de una clase y de una interfaz y no pueden usarse en los comentarios de métodos, constructores o campos. Ambas etiquetas pueden estar seguidas de cualquier texto y no se requiere ningún formato especial para ninguna de ellas. Ejemplos:

```
@author Hakcer T. LargeBrain
@version 2004.12.31
```

Las etiquetas `@param` y `@throws` se usan en métodos y en constructores, mientras que `@return` se usa sólo en métodos. Algunos ejemplos:

```
@param   limite El valor máximo permitido.
@return Un número aleatorio en el rango 1 a limite (inclusive)
@throws IllegalLimitException  Si el límite es menor que 1.
```

La etiqueta `@see` adopta varias formas diferentes y puede usarse en cualquier comentario de documentación. Proporciona un camino de referencia cruzada hacia un comentario de otra clase, método o cualquier otra forma de documentación. Se agrega una sección `See Also` al elemento que está siendo comentado. Algunos ejemplos típicos:

```
@see "The Java Language Specification, by Joy et al"
@see <a href=http://www.bluej.org/>The BlueJ web site </a>
@see #estaVivo
@see java.util.ArrayList#add
```

La primera simplemente encierra un texto en forma de cadena sin un hipervínculo, la segunda es un hipervínculo hacia el documento especificado, la tercera es un vínculo a la documentación del método `estaVivo` de la misma clase, la cuarta vincula la documentación del método `add` con la clase `java.util.ArrayList`.

I.2 Soporte de BlueJ para javadoc

Si un proyecto ha sido comentado usando el estilo de `javadoc`, BlueJ ofrece utilidades para generar la documentación HTML completa. En la ventana principal, seleccione el elemento *Tools/Project Documentation* del menú y se generará la documentación (si es necesario) y se mostrará en la ventana de un navegador.

Dentro del editor de BlueJ, se puede pasar de la vista del código fuente de una clase a la vista de su documentación cambiando la opción *Implementation* por la opción *Interface* en la parte superior derecha de la ventana (Figura I.1). Esta opción ofrece una vista previa y rápida de la documentación pero no contendrá referencias a la documentación de las superclases o de las clases que se usan.

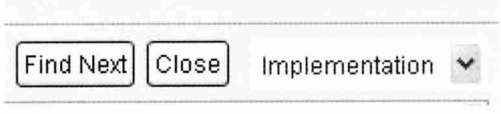


Figura I.1
La opción de vistas `Implementation` e `Interface`

Puede encontrar más detalles sobre el documentador de java en la dirección

`http://java.sun.com/j2se/javadoc/writingdoccomments/index.html`

APÉNDICE

J

Guía de estilo de programación

J.1 Nombres

J.1.1 Use nombres significativos

Use nombres descriptivos para todos los identificadores (nombres de clases, de variables, de métodos). Evite ambigüedades. Evite abreviaturas. Los métodos de modificación debieran comenzar con el prefijo «*set*»: *setAlgo(...)*. Los métodos de acceso debieran comenzar con el prefijo «*get*»: *getAlgo(...)*. Los métodos de acceso con valores de retorno booleanos generalmente comienzan con el prefijo «es»: *esAlgo(...)*; por ejemplo, `esVacio()`.

J.1.2 Los nombres de las clases comienzan con una letra mayúscula

J.1.3 Los nombres de las clases son sustantivos en singular

J.1.4 Los nombres de los métodos y de las variables comienzan con letras minúsculas

Tanto los nombres de las clases, como los de los métodos y los de las variables, emplean letras mayúsculas entre medio para aumentar la legibilidad de los identificadores que lo componen; por ejemplo: `numeroDeElementos`.

J.1.5 Las constantes se escriben en MAYÚSCULAS

Ocasionalmente se utiliza el símbolo de subrayado en el nombre de una constante para diferenciar los identificadores que lo componen: TAMANIO_MAXIMO.

J.2 Esquema

J.2.1 Un nivel de indentación es de cuatro espacios

J.2.2 Todas las sentencias de un bloque se indentan un nivel

J.2.3 Las llaves de las clases y de los métodos se ubican solas en una línea

Las llaves que encierran el bloque de código de la clase y las de los bloques de código de los métodos se escriben en una sola línea y con el mismo nivel de indentación. Por ejemplo:

```
public int getEdad()
{
      sentencias
}
```

J.2.4 Para los restantes bloques de código, las llaves se abren al final de una línea

En todos los bloques de código restantes, la llave se abre al final de la línea que contiene la palabra clave que define al bloque. La llave se cierra en una línea independiente, alineada con la palabra clave que define dicho bloque. Por ejemplo:

```
while(condición) {
      sentencias
}

if(condición) {
      sentencias
}
else {
      sentencias
}
```

J.2.5 Use siempre llaves en las estructuras de control

Se usan llaves en las sentencias *if* y en los ciclos aun cuando el cuerpo esté compuesto por una única sentencia.

J.2.6 Use un espacio antes de la llave de apertura de un bloque de una estructura de control

J.2.7 Use un espacio antes y después de un operador

J.2.8 Use una línea en blanco entre los métodos (y los constructores)

Use líneas en blanco para separar bloques lógicos de código; es decir, use líneas en blanco por lo menos entre métodos, pero también entre las partes lógicas dentro de un mismo método.

J.3 Documentación

J.3.1 Cada clase tiene un comentario de clase en su parte superior

El comentario de clase contiene como mínimo

- una descripción general de la clase

- el nombre del autor (o autores)
- un número de versión

Cada persona que ha contribuido en la clase debe ser nombrada como un autor o debe ser acreditada apropiadamente de otra manera.

Un número de versión puede ser simplemente un número o algún otro formato. Lo más importante es que el lector pueda reconocer si dos versiones no son iguales y determinar cuál es la más reciente.

J.3.2 Cada método tiene un comentario

J.3.3 Los comentarios son legibles para javadoc

Los comentarios de la clase y de los métodos deben ser reconocidos por javadoc; en otras palabras: deben comenzar con el símbolo de comentario «/**».

J.3.4 Comente el código sólo donde sea necesario

Se deben incluir comentarios en el código en los lugares en que no resulte obvio o sea difícil de comprender (y preferentemente, el código debe ser obvio o fácil de entender, siempre que sea posible) y donde ayude a la comprensión de un método. No comente sentencias obvias, ¡asuma que el lector comprende Java!

J.4 Restricciones de uso del lenguaje

J.4.1 Orden de las declaraciones: campos, constructores, métodos

Los elementos de una definición de clase aparecen (si se presentan) en el siguiente orden: sentencias de paquete, sentencias de importación, comentario de clase, encabezado de la clase, definición de campos, constructores, métodos.

J.4.2 Los campos no deben ser públicos (con excepción de los campos final)

J.4.3 Use siempre modificadores de acceso

Especifique todos los campos y los métodos como privados, públicos o protegidos. Nunca use el acceso por defecto (*package private*).

J.4.4 Importe las clases individualmente

Es preferible que las sentencias de importación nombren explícitamente cada clase que se quiere importar y no al paquete completo. Por ejemplo:

```
import java.util.ArrayList;
import java.util.HashSet;
```

es mejor que

```
import java.util.*;
```

J.4.5 Incluya siempre un constructor (aun cuando su cuerpo quede vacío)

J.4.6 Incluya siempre una llamada al constructor de una superclase

En los constructores de las subclases no deje que se realice la inserción automática de una llamada a una superclase; incluya explícitamente la invocación `super(...)`, aun cuando funcione bien sin hacerlo.

J.4.7 Inicialice todos los campos en el constructor

J.5 Modismos del código

J.5.1 Use iteradores en las colecciones

Para iterar o recorrer una colección, use un *ciclo for-each*. Cuando la colección debe ser modificada durante una iteración, use un `Iterator` en lugar de un índice entero.

APÉNDICE

K

Clases importantes de la biblioteca de Java

La plataforma Java 2 incluye un rico conjunto de bibliotecas que sustentan una amplia variedad de tareas de programación.

En este apéndice resumiremos brevemente los detalles de algunas de las clases e interfaces de los paquetes más importantes del API de la plataforma Java 2. Un programador Java competente debe estar familiarizado con la mayoría de ellas. Este apéndice es sólo un resumen y debe leerse conjuntamente con toda la documentación del API de Java.

K.1 El paquete `java.lang`

Las clases y las interfaces que contiene el paquete `java.lang` son fundamentales para el lenguaje Java; es por este motivo que este paquete se importa automática e implícitamente en cualquier definición de clase.

paquete `java.lang`	**Síntesis de las clases más importantes**
clase `Math`	`Math` es una clase que contiene sólo campos y métodos estáticos. En esta clase se definen los valores de las constantes matemáticas **e** y π, las funciones trigonométricas y otras funciones como **`abs`**, **`min`**, **`max`** y **`sqrt`** (raíz cuadrada).
clase `Object`	`Object` es la superclase de todas las clases, está en la raíz de todas las jerarquías de clases. Todos los objetos heredan de ella la implementación por defecto de métodos importantes como **`equals`** y **`toString`**. Otros métodos significativos definidos en esta clase son **`clone`** y **`hashCode`**.
clase `String`	Las cadenas constituyen una característica importante de muchas aplicaciones y reciben un tratamiento especial en Java. Los métodos más importantes de esta clase son **`charAt`**, **`equals`**, **`indexOf`**, **`length`**, **`split`** y **`substring`**. Las cadenas definidas a partir de esta clase son objetos inmutables, por lo tanto, métodos tales como **`trim`**, que parecieran ser métodos de modificación, en realidad devuelven un nuevo objeto **`String`** que representa el resultado de la operación.
clase `StringBuffer`	La clase **`StringBuffer`** aporta una alternativa eficiente a la clase **`String`**, en los casos en que se requiere construir una cadena a partir de un conjunto de componentes, como ocurre por ejemplo, en la concatenación. Sus métodos más importantes son **`append`**, **`insert`** y **`toString`**.

K.2 El paquete `java.util`

El paquete `java.util` es una colección relativamente incoherente de clases e interfaces útiles.

paquete `java.util`	Síntesis de las clases más importantes
interfaz `Collection`	Esta interfaz proporciona el conjunto central de los métodos de la mayoría de las clases basadas en colecciones, que se definen en el paquete `java.util` tales como `ArrayList`, `HashSet` y `LinkedList`. Define la signatura de los métodos `add`, `clear`, `iterator`, `remove` y `size`.
interfaz `Iterator`	`Iterator` define una interfaz sencilla y consistente para recorrer el contenido de una colección. Sus tres métodos son `hasNext`, `next` y `remove`.
interfaz `List`	`List` es una extensión de la interfaz `Collection` y proporciona medios para tratar la colección como una secuencia; por este motivo muchos de sus métodos tienen un índice como parámetro, como por ejemplo: `add`, `get`, `remove` y `set`. Clases tales como `ArrayList` y `LinkedList` implementan la interfaz `List`.
interfaz `Map`	La interfaz `Map` ofrece una alternativa a las colecciones basadas en listas mediante la idea de asociar cada objeto de una colección con un valor clave. Los objetos se agregan y se acceden mediante sus métodos `put` y `get`. Observe que un `Map` no retorna un objeto `Iterator` sino que su método `keySet` devuelve un objeto `Set` de claves y su método `values` retorna un objeto `Collection` con los objetos del mapa.
interfaz `Set`	La interfaz `Set` es una extensión de la interfaz `Collection` que tiene la intención de asignar una colección que no contenga elementos duplicados. Dado que es una interfaz, merece la pena mencionar que `Set` no está implicada realmente en reforzar esta restricción. Esto quiere decir que `Set` es en realidad una interfaz indicativa, que permite a los implementadores de colecciones indicar que sus clases cumplen con esta particular restricción.
clase `ArrayList`	Es una implementación de la interfaz `List` que usa un arreglo con el fin de proporcionar acceso directo eficiente, mediante índices, a los objetos almacenados. Si se agregan o se eliminan objetos en cualquier lugar, excepto de la posición final de la lista, se deben desplazar los siguientes elementos para hacer espacio o para tapar los agujeros que quedan. Los métodos más importantes son `add`, `get`, `iterator`, `remove` y `size`.
clase `Collections`	Esta clase reúne los métodos estáticos que se usan para manipular las colecciones. Los métodos más importantes son `binarySearch`, `fill` y `sort`.
clase `HashMap`	`HashMap` es una implementación de la interfaz `Map`. Los métodos más importantes son `get`, `put`, `remove` y `size`. Para recorrer un `HashMap` generalmente se realiza un proceso en dos etapas: primero se obtiene el conjunto de claves mediante su método `keySet` y luego se recorre este conjunto de claves.
clase `HashSet`	`HashSet` es una implementación de la interfaz `Set` basada en la técnica de *hashing*. Su uso es más parecido al de una `Collection` que al de un `HashMap`. Sus métodos más importantes son `add`, `remove` y `size`.
clase `LinkedList`	`LinkedList` es una implementación de la interfaz `List` cuya estructura interna para almacenar objetos responde a una lista simplemente enlazada. El acceso directo a los extremos de la lista es eficiente, pero no es tan eficiente el acceso a objetos individuales mediante un índice como el de un `ArrayList`. Por otro lado, al agregar objetos o al eliminarlos de la lista no es necesario hacer ningún cambio en los objetos existentes. Los métodos más importantes son `add`, `getFirst`, `getLast`, `iterator`, `removeFirst`, `removeLast` y `size`.

paquete `java.util`	Síntesis de las clases más importantes
clase `Random`	La clase `Random` colabora en la generación de valores pseudo aleatorios, los típicos números aleatorios. La secuencia de números generada está determinada por un valor semilla que puede pasarse al constructor o asignarse mediante una llamada al método `setSeed`. Dos objetos `Random` que comienzan con la misma semilla devolverán la misma secuencia de valores ante llamadas idénticas. Los métodos más importantes son `nextBoolean`, `nextDouble`, `nextInt` y `setSeed`.
clase `Scanner`	La clase `Scanner` proporciona un medio para leer y analizar entradas. Se utiliza generalmente para leer entradas desde el teclado. Los métodos más importantes son `next` y `hasNext`.

K.3 El paquete `java.io`

El paquete `java.io` contiene clases que permiten las operaciones de entrada y de salida (*input/output*). Muchas de las clases se diferencian porque se basan en flujos (*stream*) (es decir, operan sobre datos binarios), o porque operan con caracteres (`readers` y `writers`).

paquete `java.io`	Síntesis de las clases más importantes
interfaz `Serializable`	La interfaz `Serializable` es una interfaz vacía que no requiere que se escriba ningún código en la implementación de una clase. Las clases implementan esta interfaz con el fin de participar del proceso de serialización. Los objetos `Serializable` deben escribirse y leerse como un todo, desde y hacia fuentes de entrada/salida. Esto hace que el almacenamiento y la recuperación de datos persistentes sea un proceso relativamente simple en Java. Vea las clases `ObjectInputStream` y `ObjectOutputStream` para más información.
clase `BufferedReader`	Es una clase que proporciona acceso a un buffer de caracteres desde una fuente de entrada. El ingreso mediante un buffer generalmente es más eficiente que sin él, especialmente si la fuente de entrada es un archivo externo al sistema. Dado que el ingreso se hace mediante un buffer, ofrece un método `readLine` que no está disponible en la mayoría de las otras clases para procesar entradas. Los métodos más importantes son `close`, `read` y `readLine`.
clase `BufferedWriter`	Es una clase que proporciona salida de caracteres mediante un buffer. La salida mediante un buffer es más eficiente que sin él especialmente si el destino de la salida es un archivo externo al sistema. Los métodos más importantes son `close`, `flush` y `write`.
clase `File`	La clase `File` proporciona una representación en objeto de archivos y carpetas (directorios) de un sistema de archivos externo. Existen métodos para indicar si un archivo es de lectura y/o de escritura y si es un archivo o una carpeta. Se puede crear un objeto `File` mediante un archivo inexistente que será el primer paso en la creación de un archivo físico en el sistema de archivos. Los métodos más importantes son: `canRead`, `canWrite`, `createNewFile`, `createTempFile`, `getName`, `getParent`, `getPath`, `isDirectory`, `isFile` y `listFiles`.
clase `FileReader`	La clase `FileReader` se usa para abrir un archivo externo preparado para que su contenido se pueda leer mediante caracteres. Un objeto `FileReader` se pasa generalmente al constructor de otra clase lectora (tal como `BufferedReader`) en lugar de usarlo directamente. Los métodos más importantes son `close` y `read`.

paquete `java.io`	Síntesis de las clases más importantes
clase `FileWriter`	La clase `FileWriter` se usa para abrir un archivo externo preparado para grabar datos mediante caracteres. Un par de constructores determinan si se agregara a un archivo existente o si el contenido existente se descarta. Un objeto `FileWriter` generalmente se pasa al constructor de otra clase escritora (tal como `BufferedWriter`) en lugar de usarlo directamente. Los métodos más importantes son: `close`, `flush` y `write`.
clase `IOException`	Es una clase de excepción comprobada que es la raíz de la jerarquía de la mayoría de las excepciones de *input/output*.

K.4 El paquete `java.net`

El paquete `java.net` contiene clases e interfaces que soportan aplicaciones para trabajar en red. La mayoría de ellas están fuera del alcance de este libro.

paquete `java.net`	Síntesis de las clases más importantes
clase `URL`	La clase `URL` representa un *Uniform Resource Locator*, en otras palabras, proporciona una manera de describir la ubicación de algo en Internet. De hecho, también se puede usar para describir la ubicación de algo en un sistema de archivos local. La hemos incluido porque las clases de `java.io` y de `javax.swing` usan con frecuencia objetos `URL`. Los métodos más importantes son: `getContent`, `getFile`, `getHost`, `getPath` y `openStream`.

K.5 Otros paquetes importantes

Otros paquetes importantes son

```
java.awt
java.awt.event
javax.swing
javax.swing.event
```

Estas clases se usan extensamente cuando se escriben interfaces gráficas de usuario (IGU) y contienen muchas clases útiles con las que los programadores de IGU deberán familiarizarse.

APÉNDICE

L

Tabla de conversión de términos que aparecen en el CD

Español	Inglés	Qué es	Proyecto	Capítulo
asignarAula	setRoom	método	lab-classes	1
asignarHorario	setTime	método	lab-classes	1
aula	room	campo	lab-classes	1
cadenaDiaHora	timeAndDayString	parámetro	lab-classes	1
cambiarNombre	changeName	método	lab-classes	1
capacidad	capacity	campo	lab-classes	1
creditos	credits	campo	lab-classes	1
CursoDeLaboratorio	LabClass	clase	lab-classes	1
diaYHora	timeAndDay	campo	lab-classes	1
Estudiante	Student	clase	lab-classes	1
estudiantes	students	campo	lab-classes	1
getCreditos	getCredits	método	lab-classes	1
getIdEstudiante	getStudentID	método	lab-classes	1
getNombreDeUsuario	getLoginName	método	lab-classes	1
iDEstudiante	studentID	parámetro	lab-classes	1
imprimir	print	método	lab-classes	1
imprimirLista	printList	método	lab-classes	1
inscribirEstudiante	enrollStudent	método	lab-classes	1
nombreCompleto	fullName	parámetro	lab-classes	1
nombreInstructor	instructorName	parámetro	lab-classes	1
nuevoEstudiante	newStudent	parámetro	lab-classes	1
nuevoNombre	replacementName	parámetro	lab-classes	1
numeroDeAula	roomNumber	parámetro	lab-classes	1
numeroDeEstudiantes	numberOfStudents	método	lab-classes	1
numeroMaximoDeEstudiantes	maxNumbreOfStudents	parámetro	lab-classes	1
obtenerNombre	getName	método	lab-classes	1
puntosAdicionales	additionalPoints	parámetro	lab-classes	1
sumarCreditos	addCredits	método	lab-classes	1
cuadro	Picture	clase	picture	1
pared	wall	campo	picture	1
ponerBlancoYNegro	setBlackAndWhite	método	picture	1
ponerColor	setColor	método	picture	1
sol	sun	campo	picture	1
techo	roof	campo	picture	1
ventana	window	campo	picture	1

Español	Inglés	Qué es	Proyecto	Capítulo
alto	height	campo	shapes	1
ancho	width	campo	shapes	1
borrar	erase	método	shapes	1
cambiarColor	changeColor	método	shapes	1
cambiarTamanio	changeSize	método	shapes	1
Circulo	Circle	clase	shapes	1
Cuadrado	Square	clase	shapes	1
diametro	diameter	campo	shapes	1
dibujar	draw	método	shapes	1
distancia	distance	parámetro	shapes	1
esVisible	isVisible	campo	shapes	1
figuras	shapes	proyecto	shapes	1
lado	size	campo	shapes	1
moverAbajo	moveDown	método	shapes	1
moverArriba	moveUp	método	shapes	1
moverDerecha	moveRight	método	shapes	1
moverHorizontal	moveHorizontal	método	shapes	1
moverIzquierda	moveLeft	método	shapes	1
moverLentoHorizontal	slowMoveHorizontal	método	shapes	1
moverLentoVertical	slowMoveVertical	método	shapes	1
nuevoAlto	newHeight	parámetro	shapes	1
nuevoAncho	newWidth	parámetro	shapes	1
nuevoColor	newColor	parámetro	shapes	1
nuevoDiametro	newDiameter	parámetro	shapes	1
nuevoLado	newSize	parámetro	shapes	1
posicionX	xPosition	campo	shapes	1
posicionY	yPosition	campo	shapes	1
Triangulo	Triangle	clase	shapes	1
volverInvisible	makeInvisible	método	shapes	1
volverVisible	makeVisible	método	shapes	1
cantidadAReintegrar	amountToRefund	variable local	better-ticket-machine	2
maquina-de-boletos-mejorada	better-ticket-machine	proyecto	better-ticket-machine	2
reintegrarSaldo	refundBalance	método	better-ticket-machine	2
autor	author	campo	book-exercise	2
autorDelLibro	bookAuthor	parámetro	book-exercise	2
ejercicio-libro	book-exercise	proyecto	book-exercise	2
Libro	Book	clase	book-exercise	2
titulo	title	campo	book-exercise	2
tituloDelLibro	bookTitle	parámetro	book-exercise	2
cantidad	amount	parámetro	naive-ticket-machine	2
imprimirBoleto	printTicket	método	naive-ticket-machine	2
ingresarDinero	insertMoney	método	naive-ticket-machine	2
MaquinaDeBoletos	TicketMachine	clase	naive-ticket-machine	2
maquina-de-boletos-simple	naive-ticket-machine	proyecto	naive-ticket-machine	2
obtenerPrecio	getPrice	método	naive-ticket-machine	2

Español	Inglés	Qué es	Proyecto	Capítulo
obtenerSaldo	getBalance	método	naive-ticket-machine	2
obtenerTotal	getTotal	método	naive-ticket-machine	2
precio	price	campo	naive-ticket-machine	2
precioDelBoleto	ticketCost	parámetro	naive-ticket-machine	2
saldo	balance	campo	naive-ticket-machine	2
actualizarVisor	updateDisplay	método	clock-display	3
cadVisor	displayString	campo	clock-display	3
getHora	getTime	método	clock-display	3
getValor	getValue	método	clock-display	3
getValorDelVisor	getDisplayValue	método	clock-display	3
horas	hours	campo	clock-display	3
incrementar	increment	método	clock-display	3
limite	limit	campo	clock-display	3
limiteMaximo	rollOverLimit	parámetro	clock-display	3
minutos	minutes	campo	clock-display	3
nuevoValor	replacementValue	parámetro	clock-display	3
ponerEnHora	setTime	método	clock-display	3
setValor	setValue	método	clock-display	3
ticTac	timeTick	método	clock-display	3
valor	value	campo	clock-display	3
VisorDeNumeros	NumberDisplay	clase	clock-display	3
VisorDeReloj	ClockDisplay	clase	clock-display	3
visor-de-reloj	clock-display	proyecto	clock-display	3
agregarEnLista	post	método	mail-system	3
cantidad	count	variable local	mail-system	3
cantidadDeMensajes	howManyMailItems	método	mail-system	3
ClienteDeCorreo	MailClient	clase	mail-system	3
de	from	campo	mail-system	3
enviarMensaje	sendMailItem	método	mail-system	3
getDe	getFrom	método	mail-system	3
getMensajeSiguiente	getNextMailItem	método	mail-system	3
getMensajeSiguiente	getNextMailItem	método	mail-system	3
getPara	getTo	método	mail-system	3
getTexto	getMessage	método	mail-system	3
imprimirMensajeSiguiente	printNextMailItem	método	mail-system	3
mensaje	item	variable local	mail-system	3
Mensaje	MailItem	clase	mail-system	3
mensajes	items	campo	mail-system	3
para	to	campo	mail-system	3
quien	who	parámetro	mail-system	3
servidor	server	campo	mail-system	3
ServidorDeCorreo	MailServer	clase	mail-system	3
sistema-de-correo	mail-system	proyecto	mail-system	3
texto	message	campo	mail-system	3
usuario	user	campo	mail-system	3

Español	Inglés	Qué es	Proyecto	Capítulo
descripcion	description	campo	auction	4
detalles	details	variable local	auction	4
éxito	successful	variable local	auction	4
getDescripcion	getDescription	método	auction	4
getOfertaMaxima	getHighestBid	método	auction	4
getOfertante	getBidder	método	auction	4
ingresarLote	enterLot	método	auction	4
Lote	Lot	clase	auction	4
loteElegido	selectedLot	variable local	auction	4
lotes	lots	campo	auction	4
mostrarLotes	showLots	método	auction	4
numero	number	campo	auction	4
numeroDeLoteSiguiente	nextLotNumber	campo	auction	4
Oferta	Bid	clase	auction	4
ofertaMaxima	highestBid	campo	auction	4
ofertante	bidder	campo	auction	4
ofertaPara	bidFor	método	auction	4
Persona	Person	clase	auction	4
Subasta	Auction	clase	auction	4
subastas	auction	proyecto	auction	4
asociar	join	método	club	4
numeroDeSocios	numberOfMembers	método	club	4
Socio	Membership	clase	club	4
agenda1	notebook1	proyecto	notebook1	4
guardarNota	storeNote	método	notebook1	4
mostrarNota	showNote	método	notebook1	4
notas	notes	campo	notebook1	4
numeroDeNota	noteNumber	parámetro	notebook1	4
numeroDeNotas	numberOfNotes	campo	notebook1	4
agenda2	notebook2	proyecto	notebook2	4
eliminarNota	removeNote	método	notebook2	4
listarTodasLasNotas	listNotes	método	notebook2	4
administrador	manager	campo	products	4
AdministradorDeStock	StockManager	clase	products	4
agregarProducto	addProduct	método	products	4
aumentarCantidad	increaseQuantity	método	products	4
buscarProducto	findProduct	método	products	4
cantidad	quantity	campo	products	4
cantidadEnStock	numberInStock	método	products	4
getAdministrador	getManager	método	products	4
getProducto	getProduct	método	products	4
mostrarDetalles	showDetails	método	products	4
mostrarDetallesDeProductos	printProductDetails	método	products	4
Producto	Product	clase	products	4
productos	products	proyecto	products	4

Español	Inglés	Qué es	Proyecto	Capítulo
recibirProducto	delivery	método	products	4
venderProducto	sellProduct	método	products	4
venderUno	sellOne	método	products	4
AnalizadorLog	LogAnalyzer	clase	weblog-analyzer	4
analizador-weblog	weblog-analyzer	proyecto	weblog-analyzer	4
analizarPorHora	analyzeHourlyData	método	weblog-analyzer	4
ANIO	YEAR	constante	weblog-analyzer	4
archivoLog	logfile	variable local	weblog-analyzer	4
archivoURL	fileURL	variable local	weblog-analyzer	4
contadoresPorHora	hourCounts	campo	weblog-analyzer	4
crearDatosSimulados	createSimulateData	método	weblog-analyzer	4
DIA	DAY	constante	weblog-analyzer	4
entrada	entry	variable local	weblog-analyzer	4
EntradaLog	LogEntry	clase	weblog-analyzer	4
formato	format	campo	weblog-analyzer	4
getFormato	getFormat	método	weblog-analyzer	4
getHora	getHour	método	weblog-analyzer	4
getMinuto	getMinute	método	weblog-analyzer	4
hayMasDatos	hasMoreEntries	método	weblog-analyzer	4
hora	hour	variable local	weblog-analyzer	4
HORA	HOUR	constante	weblog-analyzer	4
imprimirContadoresPorHora	printHourlyCounts	método	weblog-analyzer	4
imprimirDatos	printData	método	weblog-analyzer	4
LectorDeArchivoLog	LogfileReader	clase	weblog-analyzer	4
LectorDeArchivoLog	LogfileReader	clase	weblog-analyzer	4
leeDato	dataRead	campo	weblog-analyzer	4
lineaDeDatos	dataLine	parámetro	weblog-analyzer	4
lineaLog	lineLog	variable local	weblog-analyzer	4
lineaLog	logLine	parámetro	weblog-analyzer	4
MES	MONTH	constante	weblog-analyzer	4
minimo	lowest	variable local	weblog-analyzer	4
MINUTO	MINUTE	constante	weblog-analyzer	4
nombreDeArchivo	filename	parámetro	weblog-analyzer	4
otraEntrada	otherEntry	parámetro	weblog-analyzer	4
separador	tokenizer	variable local	weblog-analyzer	4
SeparadorDeLineaLog	LogLineTokenizer	clase	weblog-analyzer	4
separar	tokenize	método	weblog-analyzer	4
siguienteEntrada	nextEntry	variable local	weblog-analyzer	4
siguienteEntrada	nextEntry	método	weblog-analyzer	4
valores	dataValues	variable local	weblog-analyzer	4
valoresPorLinea	itemsPerLine	variable local	weblog-analyzer	4
borrarCirculo	eraseCircle	método	balls	5
borrarContorno	eraseOutline	método	balls	5
borrarRectangulo	eraseRectangle	método	balls	5
borrarTexto	eraseString	método	balls	5

Español	Inglés	Qué es	Proyecto	Capítulo
colorDeFondo	backgroundColor	campo	balls	5
colorPelota	ballColor	parámetro	balls	5
cuadroCanvas	drawingCancas	parámetro	balls	5
demoDibujar	drawDemo	método	balls	5
desaceleracionPelota	ballDegradatio	campo	balls	5
diametroPelota	ballDiameter	parámetro	balls	5
dibujarImagen	drawImage	método	balls	5
dibujarLinea	drawLine	método	balls	5
dibujarTexto	drawString	método	balls	5
espera	wait	método	balls	5
fdColor	bgColor	parámetro	balls	5
figura	shape	parámetro	balls	5
getColorDeFondo	getBackgroundColor	método	balls	5
getColorDeLapiz	getForegroundColor	método	balls	5
getTipoDeLetra	getFont	método	balls	5
grafico	graphic	campo	balls	5
GRAVEDAD	GRAVITY	constante	balls	5
imagenDelCanvas	canvasImage	campo	balls	5
imagenVieja	oldImage	variable local	balls	5
miCanvas	myCanvas	campo	balls	5
milisegundos	milliseconds	parámetro	balls	5
mover	move	método	balls	5
nuevoTipoDeLetra	newFont	parámetro	balls	5
pelotas	balls	proyecto	balls	5
PelotasDemo	BallDemo	clase	balls	5
piso	ground	variable local	balls	5
posicionDelPiso	groundPosition	campo	balls	5
posPiso	groundPos	parámetro	balls	5
rebotar	bounce	método	balls	5
ReboteDePelota	BouncingBall	clase	balls	5
rellenar	fill	método	balls	5
rellenarCirculo	fillCircle	método	balls	5
rellenarRectangulo	fillRectangle	método	balls	5
setColorDeFondo	setBackgroundColor	método	balls	5
setColorDeLapiz	setForegroundColor	método	balls	5
setTamanio	setSize	método	balls	5
setTipoDeLetra	setFont	método	balls	5
tamanio	size	variable local	balls	5
velocidadY	ySpeed	campo	balls	5
Contestador	Responder	clase	tech-support1	5
entrada	input	variable local	tech-support1	5
generarRespuesta	generateResponse	método	tech-support1	5
getEntrada	getInput	método	tech-support1	5
imprimirBienvenida	printWelcome	método	tech-support1	5
imprimirDespedida	printGoodBye	método	tech-support1	5

Español	Inglés	Qué es	Proyecto	Capítulo
iniciar	start	método	tech-support1	5
lector	reader	campo	tech-support1	5
LectorDeEntrada	InputReader	clase	tech-support1	5
linea	inputLine	variable local	tech-support1	5
respuesta	response	variable local	tech-support1	5
SistemaDeSoporte	SupportSystem	clase	tech-support1	5
soporte-tecnico1	tech-support1	proyecto	tech-support1	5
terminado	finished	variable local	tech-support1	5
generadorDeAzar	randomGenerator	campo	tech-support2	5
indice	index	variable local	tech-support2	5
rellenarRespuestas	fillResponses	método	tech-support2	5
respuestas	responses	campo	tech-support2	5
soporte-tecnico2	tech-support2	proyecto	tech-support2	5
arregloDePalabras	wordArray	variable local	tech-support-complete	5
contestador	responder	campo	tech-support-complete	5
imprimirDespedida	printGoodBye	método	tech-support-complete	5
lector	reader	campo	tech-support-complete	5
mapaDeRespuestas	responseMap	campo	tech-support-complete	5
rellenarMapaDeRespuetas	fillResponseMap	método	tech-support-complete	5
rellenarRespuestasPorDefecto	fillDeafultResponses	método	tech-support-complete	5
respuestasPorDefecto	defaultResponses	método	tech-support-complete	5
soporte-tecnico-completo	tech-support-complete	proyecto	tech-support-complete	5
tomarRespuestaPorDefecto	pickDefaultResponse	método	tech-support-complete	5
ALTO_BASE	BASE_HEIGHT	constante	bricks	6
cara1	side1	variable local	bricks	6
getPeso	getWeight	método	bricks	6
getSuperficieTotal	getSurfaceArea	método	bricks	6
getVolumen	getVolume	método	bricks	6
Ladrillo	Brick	clase	bricks	6
ladrillos	bricks	proyecto	bricks	6
ladrillosEnPlano	bricksInPlane	campo	bricks	6
numeroDeLadrillos	numberOfBricks	variable local	bricks	6
Pallete	Pallet	clase	bricks	6
PESO_BASE	BASE_WEIGHT	constante	bricks	6
PESO_POR_CM3	WEIGHT_PER_CM3	constante	bricks	6
unLadrillo	aBrick	campo	bricks	6
aplicarOperadorPrevio	applyPreviousOperator	método	calculator-engine	6
calculadora-motor	calculator-engine	proyecto	calculator-engine	6
getAutor	getAuthor	método	calculator-engine	6
getTitulo	getTitle	método	calculator-engine	6
getValorEnVisor	getDisplayValue	método	calculator-engine	6
igual	equals	método	calculator-engine	6
limpiar	clear	método	calculator-engine	6
mas	plus	método	calculator-engine	6
menos	minus	método	calculator-engine	6

Español	Inglés	Qué es	Proyecto	Capítulo
motor	engine	campo	calculator-engine	6
MotorDeCalculadora	CalcEngine	clase	calculator-engine	6
MotorDeCalculadoraProbador	CalcEngineTester	clase	calculator-engine	6
numeroPresionado	numberPressed	método	calculator-engine	6
operadorPrevio	previousOperator	campo	calculator-engine	6
operandoIzquierdo	leftOperand	campo	calculator-engine	6
testMas	testPlus	método	calculator-engine	6
testMenos	testMinus	método	calculator-engine	6
valorEnVisor	displayValue	campo	calculator-engine	6
calculadora-motor-impresión	calculator-engine-print	proyecto	calculator-engine-print	6
donde	where	parámetro	calculator-engine-print	6
informarEstado	reportState	método	calculator-engine-print	6
aplicarOperador	applyOperator	método	calculator-full-solution	6
calculadora-solucion-completa	calculator-full-solution	proyecto	calculator-full-solution	6
calcularResultado	calculateResult	método	calculator-full-solution	6
construyeValorEnVisor	buildingDisplayValue	campo	calculator-full-solution	6
errorEnSecuenciaDeTeclas	keySequenceError	método	calculator-full-solution	6
tieneOperandoIzquierd	haveLeftOperand	campo	calculator-full-solution	6
ultimoOperador	lastOperator	campo	calculator-full-solution	6
actualizarVisor	redisplay	método	calculator-gui	6
agregarBoton	addButton	método	calculator-gui	6
armarFrame	makeFrame	método	calculator-gui	6
Calculadora	Calculator	clase	calculator-gui	6
calculadora-igu	calculator-gui	proyecto	calculator-gui	6
estado	status	campo	calculator-gui	6
InterfazDeUsuario	UserInterface	clase	calculator-gui	6
mostrarInformacion	showInfo	método	calculator-gui	6
muestraAutor	showingAuthor	campo	calculator-gui	6
textoDelBoton	buttonText	parámetro	calculator-gui	6
visor	display	campo	calculator-gui	6
agenda-diaria-prototipo	diary-prototipe	proyecto	diary-prototype	6
anotarCita	makeAppointment	método	diary-prototype	6
buscarEspacio	findSpace	método	diary-prototype	6
cantidad_filas_requeridas	further_slots_required	variable local	diary-prototype	6
Cita	Appointment	clase	diary-prototype	6
citas	appointments	campo	diary-prototype	6
Dia	Day	clase	diary-prototype	6
diaEnAnio	dayInYear	variable local	diary-prototype	6
diaEnSemana	dayInWeek	variable local	diary-prototype	6
diaNumero	dayNumber	campo	diary-prototype	6
dias	days	campo	diary-prototype	6
DIAS_AGENDABLES_POR_SEMANA	BOOKABLE_DAYS_PER_WEEK	constante	diary-prototype	6
duracion	duration	campo	diary-prototype	6
fila	slot	variable local	diary-prototype	6

Español	Inglés	Qué es	Proyecto	Capítulo
filaSiguiente	nextSlot	variable local	diary-prototype	6
getCita	getAppointment	método	diary-prototype	6
getDia	getDay	método	diary-prototype	6
getDiaNumero	getDayNumber	método	diary-prototype	6
getDuracion	getDuration	método	diary-prototype	6
getSemanaNumero	getWeekNumber	método	diary-prototype	6
horaInicio	startTime	variable local	diary-prototype	6
horaValida	validTime	variable local	diary-prototype	6
MAX_CITAS_POR_DIA	MAX_APPOINTMENTS_PER_DAY	constante	diary-prototype	6
mostrarCitas	showAppointments	método	diary-prototype	6
PRIMER_HORA	START_OF_DAY	constante	diary-prototype	6
Semana	Week	clase	diary-prototype	6
semanaNumero	weekNumber	campo	diary-prototype	6
ULTIMA_HORA	FINAL_APPOINTMENT_TIME	constante	diary-prototype	6
agenda-diaria-prueba	diary-testing	proyecto	diary-testing	6
anotarTresCitas	makeThreeAppointments	método	diary-testing	6
citaMala	badAppointment	variable local	diary-testing	6
completarElDia	fillTheDay	método	diary-testing	6
primera	first	variable local	diary-testing	6
probarDobleCita	testDoubleBooking	método	diary-testing	6
PruebaUnaHora	OneHourTests	clase	diary-testing	6
segunda	second	variable local	diary-testing	6
tercera	third	variable local	diary-testing	6
agenda-diaria-prueba-junit-v1	diary-testing-junit-v1	proyecto	diary-testing-junit-v1	6
DiaTest	DayTest	clase	diary-testing-junit-v1	6
testAnotarTresCitas	testMakeThreeAppointments	método	diary-testing-junit-v1	6
testDobleCita	testDoubleBooking	método	diary-testing-junit-v1	6
Analizador	Parser	clase	zuul-bad	7
bar	pub	variable local	zuul-bad	7
Comando	Command	clase	zuul-bad	7
comandos	commands	campo	zuul-bad	7
comandosValidos	validCommands	método	zuul-bad	7
crearHabitaciones	createRooms	método	zuul-bad	7
esComando	isCommand	método	zuul-bad	7
esDesconocido	isUnknown	método	zuul-bad	7
establecerSalidas	setExits	método	zuul-bad	7
este	east	parámetro	zuul-bad	7
exterior	outside	variable local	zuul-bad	7
getComando	getCommand	método	zuul-bad	7
getPalabraComando	getCommandWord	método	zuul-bad	7
getSegundaPalabra	getSecondWord	método	zuul-bad	7
Habitacion	Room	clase	zuul-bad	7
habitacionActual	currentRoom	campo	zuul-bad	7

Español	Inglés	Qué es	Proyecto	Capítulo
imprimirAyuda	printHelp	método	zuul-bad	7
imprimirBienvenida	printWelcome	método	zuul-bad	7
irAHabitacion	goRoom	método	zuul-bad	7
Juego	Game	clase	zuul-bad	7
jugar	play	método	zuul-bad	7
laboratorio	lab	variable local	zuul-bad	7
lector	reader	campo	zuul-bad	7
lineaIngresada	inputLine	variable local	zuul-bad	7
norte	north	parámetro	zuul-bad	7
oeste	west	parámetro	zuul-bad	7
oficina	office	variable local	zuul-bad	7
palabraComando	commandWord	campo	zuul-bad	7
PalabrasComando	CommandWords	clase	zuul-bad	7
primerPalabra	firstWord	parámetro	zuul-bad	7
procesarComando	processCommand	método	zuul-bad	7
quiereSalir	wantToQuit	variable local	zuul-bad	7
salidaEste	eastExit	campo	zuul-bad	7
salidaNorte	northExit	campo	zuul-bad	7
salidaOeste	westExit	campo	zuul-bad	7
salidaSur	southExit	campo	zuul-bad	7
salir	quit	método	zuul-bad	7
segundaPalabra	secondWord	campo	zuul-bad	7
separador	tokenizer	variable local	zuul-bad	7
siguienteHabitacion	nextRoom	variable local	zuul-bad	7
sur	south	parámetro	zuul-bad	7
teatro	theatre	variable local	zuul-bad	7
tieneSegundaPalabra	hasSecondWord	método	zuul-bad	7
unaCadena	aString	parámetro	zuul-bad	7
zuul-malo	zull-bad	proyecto	zuul-bad	7
establecerSalida	setExit	método	zuul-better	7
getDescripcionCorta	getShortDescription	método	zuul-better	7
getDescripcionLarga	getLongDescription	método	zuul-better	7
getSalida	getExit	método	zuul-better	7
getStringDeSalidas	getExitString	método	zuul-better	7
llaves	keys	variable local	zuul-better	7
mostrarComandos	showCommands	método	zuul-better	7
mostrarTodos	showAll	método	zuul-better	7
salidas	exits	campo	zuul-better	7
stringADevolver	stringResult	variable local	zuul-better	7
vecina	neighbor	parámetro	zuul-better	7
zuul-mejorado	zuul-better	proyecto	zuul-better	7
AYUDA	HELP	valor	zuul-with-enums-v1	7
DESCONOCIDA	UNKNOWN	valor	zuul-with-enums-v1	7
getPalabraComando	getCommandWord	método	zuul-with-enums-v1	7
IR	GO	valor	zuul-with-enums-v1	7

Español	Inglés	Qué es	Proyecto	Capítulo
PalabraComando	CommandWord	clase	zuul-with-enums-v1	7
SALIR	QUIT	valor	zuul-with-enums-v1	7
zuul-con-enumeraciones-v1	zuul-with-enums-v1	proyecto	zuul-with-enums-v1	7
cadenaComando	commandString	campo	zuul-with-enums-v2	7
zuul-con-enumeraciones-v2	zuul-with-enums-v2	proyecto	zuul-with-enums-v2	7
agregarCD	addCD	método	dome-v1	8
agregarDVD	addDVD	método	dome-v1	8
BaseDeDatos	DataBase	clase	dome-v1	8
comentario	comment	campo	dome-v1	8
elArtista	theArtist	parámetro	dome-v1	8
elCD	theCD	parámetro	dome-v1	8
elDVD	theDVD	parámetro	dome-v1	8
elTitulo	theTitle	parámetro	dome-v1	8
getLoTengo	getOwn	método	dome-v1	8
interprete	artist	campo	dome-v1	8
listar	list	método	dome-v1	8
loTengo	gotIt	campo	dome-v1	8
mePertenece	ownIt	parámetro	dome-v1	8
numeroDeTemas	numberOfTracks	campo	dome-v1	8
setLoTengo	setOwn	método	dome-v1	8
temas	time	parámetro	dome-v1	8
tiempo	tracks	parámetro	dome-v1	8
tiempoDuracion	playingTime	campo	dome-v1	8
agregarElemento	addItem	método	dome-v2	8
elElemento	theItem	parámetro	dome-v2	8
Elemento	Item	clase	dome-v2	8
elementoSalir	items	campo	dome-v2	8
adyacente	adjacent	variable local	foxes-and-rabbits-v1	10
altoGrilla	gridHeight	variable local	foxes-and-rabbits-v1	10
ANCHO_POR_DEFECTO	DEFAULT_WIDTH	constante	foxes-and-rabbits-v1	10
anchoGrilla	gridWidth	variable local	foxes-and-rabbits-v1	10
buscarComida	findFood	método	foxes-and-rabbits-v1	10
Campo	Field	clase	foxes-and-rabbits-v1	10
campoActual	currentField	parámetro	foxes-and-rabbits-v1	10
campoActualizado	updatedField	parámetro	foxes-and-rabbits-v1	10
campoImage	fieldImage	variable local	foxes-and-rabbits-v1	10
cazar	hunt	método	foxes-and-rabbits-v1	10
claseDeAnimal	animalClass	parámetro	foxes-and-rabbits-v1	10
COLOR_DESCONOCIDO	UNKNOWN_COLOR	constante	foxes-and-rabbits-v1	10
COLOR_VACIA	EMPTY_COLOR	constante	foxes-and-rabbits-v1	10
colores	colors	campo	foxes-and-rabbits-v1	10
colSiguiente	nextCol	variable local	foxes-and-rabbits-v1	10
columnas	coffset	variable local	foxes-and-rabbits-v1	10
Conejo	Rabbit	clase	foxes-and-rabbits-v1	10
conejosNuevos	newRabbits	parámetro	foxes-and-rabbits-v1	10

Español	Inglés	Qué es	Proyecto	Capítulo
Contador	Counter	clase	foxes-and-rabbits-v1	10
contadores	counters	campo	foxes-and-rabbits-v1	10
correr	run	método	foxes-and-rabbits-v1	10
cuentasValidas	countsValid	campo	foxes-and-rabbits-v1	10
dibujarMarca	drawMark	método	foxes-and-rabbits-v1	10
direccionAdyacenteLibre	freeAdjacentLocation	método	foxes-and-rabbits-v1	10
direccionAdyacentePorAzar	randomAdjacentLocatio	método	foxes-and-rabbits-v1	10
direccionesAdyacentes	adjacentLocations	variable local	foxes-and-rabbits-v1	10
edad	age	campo	foxes-and-rabbits-v1	10
EDAD_DE_REPRODUCCION	BREEDING_AGE	constante	foxes-and-rabbits-v1	10
EDAD_MAX	MAX_AGE	constante	foxes-and-rabbits-v1	10
edadPorAzar	randomAge	parámetro	foxes-and-rabbits-v1	10
ejecutarSimulacionLarga	runLongSimulation	método	foxes-and-rabbits-v1	10
estadisticas	stats	campo	foxes-and-rabbits-v1	10
EstadisticasDelCampo	FieldStats	clase	foxes-and-rabbits-v1	10
estaVivo	isAlive	método	foxes-and-rabbits-v1	10
esViable	isViable	método	foxes-and-rabbits-v1	10
etiquetaPaso	stepLabel	campo	foxes-and-rabbits-v1	10
FACTOR_DE_ESCALA_DEL_VISOR_DE_GRILLA	GRID_VIEW_SCALING_FACTOR	constante	foxes-and-rabbits-v1	10
filas	roffset	variable local	foxes-and-rabbits-v1	10
filaSiguiente	nextRow	variable local	foxes-and-rabbits-v1	10
generarCuentas	generateCounts	método	foxes-and-rabbits-v1	10
getCantidad	getCount	método	foxes-and-rabbits-v1	10
getDetallesDePoblacion	getPopulationDetails	método	foxes-and-rabbits-v1	10
getFila	getRow	método	foxes-and-rabbits-v1	10
incrementarContador	incrementCount	método	foxes-and-rabbits-v1	10
incrementarEdad	incrementAge	método	foxes-and-rabbits-v1	10
incrementarHambre	incrementHunger	método	foxes-and-rabbits-v1	10
LARGO_POR_DEFECTO	DEFAULT_DEPTH	constante	foxes-and-rabbits-v1	10
limpiar	clear	método	foxes-and-rabbits-v1	10
llave	key	variable local	foxes-and-rabbits-v1	10
lugar	where	variable local	foxes-and-rabbits-v1	10
MAX_TAMANIO_DE_CAMADA	MAX_LITTER_SIZE	constante	foxes-and-rabbits-v1	10
mostrarEstado	showStatus	método	foxes-and-rabbits-v1	10
nacimientos	births	variable local	foxes-and-rabbits-v1	10
nivelDeComida	foodLevel	campo	foxes-and-rabbits-v1	10
nuevaUbicacion	newLocation	variable local	foxes-and-rabbits-v1	10
nuevoConejo	newRabbit	variable local	foxes-and-rabbits-v1	10
nuevoZorro	newFox	variable local	foxes-and-rabbits-v1	10
numeroDePasos	numSteps	parámetro	foxes-and-rabbits-v1	10
otra	other	variable local	foxes-and-rabbits-v1	10
poblacion	population	campo	foxes-and-rabbits-v1	10
poblar	populate	método	foxes-and-rabbits-v1	10

Español	Inglés	Qué es	Proyecto	Capítulo
PREFIJO_PASO	STEP_PREFIX	constante	foxes-and-rabbits-v1	10
PREFIJO_POBLACION	POPULATION_PREFIX	constante	foxes-and-rabbits-v1	10
PROBABILIDAD_DE_ CREACION_DEL_CONEJO	RABBIT_CREATION_ PROBABILITY	constante	foxes-and-rabbits-v1	10
PROBABILIDAD_DE_ CREACION_DEL_ZORRO	FOX_CREATION_ PROBABILITY	constante	foxes-and-rabbits-v1	10
PROBABILIDAD_DE_ REPRODUCCION	BREEDING_ PROBABILITY	constante	foxes-and-rabbits-v1	10
reinicializar	reset	método	foxes-and-rabbits-v1	10
reproducir	breed	método	foxes-and-rabbits-v1	10
sePuedeReproducir	canBreed	método	foxes-and-rabbits-v1	10
setComido	setEaten	método	foxes-and-rabbits-v1	10
setUbicacion	setLocation	método	foxes-and-rabbits-v1	10
Simulador	Simulator	clase	foxes-and-rabbits-v1	10
simular	simulate	método	foxes-and-rabbits-v1	10
simularUnPaso	simulateOneStep	método	foxes-and-rabbits-v1	10
terminoCuenta	countFinished	método	foxes-and-rabbits-v1	10
ubi	loc	variable local	foxes-and-rabbits-v1	10
Ubicacion	Location	clase	foxes-and-rabbits-v1	10
ubicación	Location	campo	foxes-and-rabbits-v1	10
ubicar	place	método	foxes-and-rabbits-v1	10
VALOR_COMIDA_CONEJO	RABBIT_FOOD_VALUE	constante	foxes-and-rabbits-v1	10
visor	view	campo	foxes-and-rabbits-v1	10
visorDelCampo	fieldView	campo	foxes-and-rabbits-v1	10
VisorDelCampo	fieldView	clase	foxes-and-rabbits-v1	10
VisorDelSimulador	SimulatorView	clase	foxes-and-rabbits-v1	10
vive	alive	campo	foxes-and-rabbits-v1	10
Zorro	Fox	clase	foxes-and-rabbits-v1	10
zorrosNuevos	newFoxes	parámetro	foxes-and-rabbits-v1	10
zorros-y-conejos-v1	foxes-and-rabbits-v1	proyecto	foxes-and-rabbits-v1	10
actuar	act	método	foxes-and-rabbits-v2	10
animalesNuevos	newAnimals	parámetro	foxes-and-rabbits-v2	10
getAnimalEn	getAnimalAt	método	foxes-and-rabbits-v2	10
setMuerto	setDead	método	foxes-and-rabbits-v2	10
zorros-y-conejos-v2	foxes-and-rabbits-v2	proyecto	foxes-and-rabbits-v2	10
construirVentana	makeFrame	método	imageviewer0-1	11
etiqueta	label	variable local	imageviewer0-1	11
panelContenedor	contentPane	variable local	imageviewer0-1	11
ventana	frame	campo	imageviewer0-1	11
VisorDeImagen	ImageViewer	clase	imageviewer0-1	11
visor-de-imagen-0-1	imageviewer0-1	proyecto	imageviewer0-1	11
barraDeMenu	menubar	variable local	imageviewer0-2	11
construirBarraDeMenu	makeMenuBar	método	imageviewer0-2	11
elementoAbrir	openItem	variable local	imageviewer0-2	11
elementoSalir	quitItem	variable local	imageviewer0-2	11

Español	Inglés	Qué es	Proyecto	Capítulo
evento	event	parámetro	imageviewer0-2	11
menuArchivo	fileMenu	variable local	imageviewer0-2	11
visor-de-imagen-0-2	imageviewer0-2	proyecto	imageviewer0-2	11
visor-de-imagen-0-3	imageviewer0-3	proyecto	imageviewer0-3	11
abrirArchivo	openFile	método	imageviewer0-4	11
AdministradorDeArchivos	ImageFileManager	clase	imageviewer0-4	11
alto	height	variable local	imageviewer0-4	11
ancho	width	variable local	imageviewer0-4	11
archivo	file	parámetro	imageviewer0-4	11
archivoDeImagen	imageFile	parámetro	imageviewer0-4	11
archivoSeleccionado	selectedFile	variable local	imageviewer0-4	11
cargarImagen	loadImage	método	imageviewer0-4	11
FORMATO_DE_IMAGEN	IMAGE_FORMAT	constante	imageviewer0-4	11
getImagen	getImage	método	imageviewer0-4	11
grabarImagen	saveImage	método	imageviewer0-4	11
graficoDeImagen	imageGraphics	variable local	imageviewer0-4	11
imagen	image	parámetro	imageviewer0-4	11
ImagenOF	OFImage	clase	imageviewer0-4	11
limpiarImagen	clearImage	método	imageviewer0-4	11
PanelDeImagen	ImagePanel	clase	imageviewer0-4	11
salir	quit	método	imageviewer0-4	11
selectorDeArchivos	fileChooser	campo	imageviewer0-4	11
setImagen	setImage	método	imageviewer0-4	11
tamanio	size	variable local	imageviewer0-4	11
valorDeRetorno	returnVal	variable local	imageviewer0-4	11
aplicarClaro	makeLighter	método	imageviewer1-0	11
aplicarOscuro	makeDarker	método	imageviewer1-0	11
aplicarUmbral	threshold	método	imageviewer1-0	11
brillo	brightness	variable local	imageviewer1-0	11
cerrar	close	método	imageviewer1-0	11
claro	lighter	método	imageviewer1-0	11
elemento	item	variable local	imageviewer1-0	11
etiquetaNombreDeArchivo	filenameLabel	campo	imageviewer1-0	11
mostrarAcercaDe	showAbout	método	imageviewer1-0	11
mostrarEstado	showStatus	método	imageviewer1-0	11
mostrarNombreDeArchivo	showFilename	método	imageviewer1-0	11
oscuro	darker	método	imageviewer1-0	11
texto	text	variable local	imageviewer1-0	11
umbral	threshold	método	imageviewer1-0	11
aplicar	apply	método	imageviewer2-0	11
aplicarFiltro	applyFilter	método	imageviewer2-0	11
crearFiltros	createFilters	método	imageviewer2-0	11
Filtro	Filter	clase	imageviewer2-0	11
FiltroClaro	LighterFilter	clase	imageviewer2-0	11
FiltroOscuro	DarkerFilter	clase	imageviewer2-0	11

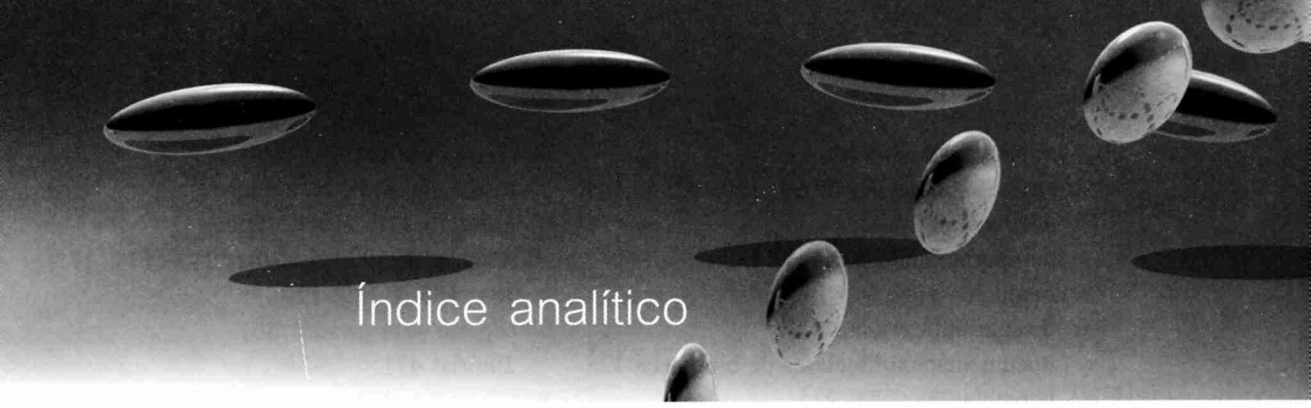
Español	Inglés	Qué es	Proyecto	Capítulo
filtros	filters	campo	imageviewer2-0	11
FiltroUmbral	ThresholdFilter	clase	imageviewer2-0	11
getNombre	getName	método	imageviewer2-0	11
listaDeFiltros	filterList	variable local	imageviewer2-0	11
achicar	makeSmaller	método	imageviewer3-0	11
agrandar	makeLarger	método	imageviewer3-0	11
arregloCalculoDeX	computeXArray	método	imageviewer3-0	11
arregloCalculoDeY	computeyArray	método	imageviewer3-0	11
arregloX	xArray	variable local	imageviewer3-0	11
arregloY	yArray	variable local	imageviewer3-0	11
barraDeHerramientas	toolbar	variable local	imageviewer3-0	11
botonAchicar	smallerButton	campo	imageviewer3-0	11
botonAgrandar	largerButton	campo	imageviewer3-0	11
DOS_PI	TWO_PI	constante	imageviewer3-0	11
ESCALA	SCALE	constante	imageviewer3-0	11
FiltroOjoDePez	FishEyeFilter	clase	imageviewer3-0	11
grabarComo	saveAs	método	imageviewer3-0	11
habilitarBotones	setButtonsEnabled	método	imageviewer3-0	11
nuevaImagen	newImage	variable local	imageviewer3-0	11
azul	blue	variable local	imageviewer-final	11
borde	edge	método	imageviewer-final	11
difAzul	diffBlue	método	imageviewer-final	11
diferencia	difference	variable local	imageviewer-final	11
difRojo	diffRed	método	imageviewer-final	11
difVerde	diffGreen	método	imageviewer-final	11
FiltroBordes	EdgeFilter	clase	imageviewer-final	11
FiltroInvertir	InvertFilter	clase	imageviewer-final	11
FiltroSuavizar	SmoothFilter	clase	imageviewer-final	11
izquierda	left	variable local	imageviewer-final	11
prom	avg	variable local	imageviewer-final	11
promAzul	avgBlue	método	imageviewer-final	11
promRojo	avgRed	método	imageviewer-final	11
promVerde	avgGreen	método	imageviewer-final	11
rojo	red	variable local	imageviewer-final	11
suavizado	smooth	método	imageviewer-final	11
TAMANIO_DEL_PIXEL	PIXEL_SIZE	constante	imageviewer-final	11
verde	green	variable local	imageviewer-final	11
archivoDeSonido	soundFile	parámetro	simplesound	11
archivosDeAudio	audioFiles	parámetro	simplesound	11
archivosLista	fileList	campo	simplesound	11
buscar	seek	método	simplesound	11
buscarArchivos	findFiles	método	simplesound	11
cargarSonido	loadSound	método	simplesound	11
comenzar	start	método	simplesound	11
construirVentana	makeFrame	método	simplesound	11

Español	Inglés	Qué es	Proyecto	Capítulo
detener	stop	método	simplesound	11
duracionActualDelSonido	currentSoundDuration	campo	simplesound	11
ejecutar	play	método	simplesound	11
ejecutar	play	método	simplesound	11
etiquetaInfo	infoLabel	campo	simplesound	11
largoActualDelSonido	currentSoundFrameLength	variable local	simplesound	11
mostrarAcercaDe	showAbout	método	simplesound	11
mostrarInformacion	showInfo	método	simplesound	11
MotorDeSonido	SoundEngine	clase	simplesound	11
nombreDeArchivo	fileName	variable local	simplesound	11
nombreDir	nameDir	parámetro	simplesound	11
nombresDeArchivosDeAudio	audioFileNames	variable local	simplesound	11
pausar	pause	método	simplesound	11
posicionABuscar	seekPosition	variable local	simplesound	11
reproductor	player	campo	simplesound	11
ReproductorDeSonidoIGU	SoundPlayerGUI	clase	simplesound	11
resumir	resume	método	simplesound	11
seleccionado	selected	variable local	simplesound	11
setVolumen	setVolume	método	simplesound	11
sonidoActual	currentSoundClip	campo	simplesound	11
sonidossimples	simplesound	proyecto	simplesound	11
sufijo	suffix	parámetro	simplesound	11
todosLosArchivos	allFiles	variable local	simplesound	11
buscarNuevamente	research	método	address-book-v1g	12
cadenaDeBusqueda	searchString	variable local	address-book-v1g	12
campoBuscar	searchField	variable local	address-book-v1g	12
campoNombre	nameField	variable local	address-book-v1g	12
camposDeUnaLinea	singleLineFields	variable local	address-book-v1g	12
campoTelefono	phoneField	variable local	address-book-v1g	12
ejecutar	run	método	address-book-v1g	12
getTamanioPreferido	getPreferredSize	método	address-book-v1g	12
LibretaDeDireccionesIGU	AddressBookGUI	clase	address-book-v1g	12
listaDeResultados	resultList	variable local	address-book-v1g	12
mostrarVentana	showWindow	método	address-book-v1g	12
panelDeDatos	detailsPanel	variable local	address-book-v1g	12
panelTabulado	tabbedArea	variable local	address-book-v1g	12
prepararZonaDeIngreso	setupDataEntry	método	address-book-v1g	12
prepararZonaDeLista	setupListArea	método	address-book-v1g	12
zonaBuscar	searchArea	variable local	address-book-v1g	12
zonaDeBotones	buttonArea	variable local	address-book-v1g	12
zonaDeDesplazamiento	scrollArea	variable local	address-book-v1g	12
zonaDeEtiquetaBuscar	searchLabelArea	variable local	address-book-v1g	12
zonaDeEtiquetaDireccion	AddressLabelArea	variable local	address-book-v1g	12
zonaDeEtiquetaNombre	nameLabelArea	variable local	address-book-v1g	12
zonaDeEtiquetaTelefono	phoneLabelArea	variable local	address-book-v1g	12

Español	Inglés	Qué es	Proyecto	Capítulo
zonaDeLista	listArea	variable local	address-book-v1g	12
zonaDireccion	addressArea	variable local	address-book-v1g	12
zonaNombre	nameArea	variable local	address-book-v1g	12
zonaTelefono	phoneArea	variable local	address-book-v1g	12
agregar	add	método	address-book-v1t	12
agregarContacto	addDetails	método	address-book-v1t	12
Analizador	Parser	clase	address-book-v1t	12
ayuda	help	método	address-book-v1t	12
buscar	search	método	address-book-v1t	12
clave	key	parámetro	address-book-v1t	12
claveEnUso	keyInUse	método	address-book-v1t	12
claveVieja	oldKey	parámetro	address-book-v1t	12
codigo	code	variable local	address-book-v1t	12
coincidencias	matches	variable local	address-book-v1t	12
comando	command	variable local	address-book-v1t	12
comandos	commands	campo	address-book-v1t	12
comandosValidos	validCommands	campo	address-book-v1t	12
comparacion	comparison	variable local	address-book-v1t	12
contacto	details	parámetro	address-book-v1t	12
contactosDeEjemplo	sampleDetails	variable local	address-book-v1t	12
contactosOrdenados	sortedDetails	variable local	address-book-v1t	12
DatosDelContacto	ContactDetails	clase	address-book-v1t	12
direccion	address	campo	address-book-v1t	12
eliminarContacto	removeDetails	método	address-book-v1t	12
encontrar	find	método	address-book-v1t	12
esComando	isCommand	método	address-book-v1t	12
finDeBusqueda	endOfSearch	variable local	address-book-v1t	12
getComando	getCommand	método	address-book-v1t	12
getContacto	getDetails	método	address-book-v1t	12
getDireccion	getAddress	método	address-book-v1t	12
getLibreta	getBook	método	address-book-v1t	12
getNombre	getName	método	address-book-v1t	12
getNumeroDeEntradas	getNumberOfEntries	método	address-book-v1t	12
getTelefono	getPhone	método	address-book-v1t	12
interaccion	interaction	campo	address-book-v1t	12
lector	reader	campo	address-book-v1t	12
leerLinea	readLine	método	address-book-v1t	12
libreta	book	campo	address-book-v1t	12
LibretaDeDirecciones	AddressBook	clase	address-book-v1t	12
LibretaDeDireccionesDemo	AddressBookDemo	clase	address-book-v1t	12
LibretaDeDireccionesInterfazDeTexto	AddressBookTextInterface	clase	address-book-v1t	12
libreta-de-direcciones-v1t	address-book-v1t	proyecto	address-book-v1t	12
listar	list	método	address-book-v1t	12
listarContactos	listDetails	método	address-book-v1t	12
modificarContacto	changeDetails	método	address-book-v1t	12

Español	Inglés	Qué es	Proyecto	Capítulo
mostrarComandos	showCommands	método	address-book-v1t	12
mostrarInterfaz	showInterface	método	address-book-v1t	12
mostrarTodos	showAll	método	address-book-v1t	12
nombre	name	campo	address-book-v1t	12
numeroDeEntradas	numberOfEntries	campo	address-book-v1t	12
otro	other	parámetro	address-book-v1t	12
otroContacto	otherDetails	variable local	address-book-v1t	12
palabra	word	variable local	address-book-v1t	12
PalabrasComando	CommandWords	clase	address-book-v1t	12
prefijo	keyPrefix	parámetro	address-book-v1t	12
resultados	results	variable local	address-book-v1t	12
telefono	phone	campo	address-book-v1t	12
todasLasEntradas	allEntries	variable local	address-book-v1t	12
getClave	getKey	método	address-book-v3t	12
NoCoincideContactoException	NoMatchingDetailsException	clase	address-book-v3t	12
arriboADestino	arrivedAtDestination	método	taxi-company-outline	14
arriboASalida	arrivedAtPickup	método	taxi-company-outline	14
asignarVehiculo	scheduleVehicle	método	taxi-company-outline	14
compania	company	parámetro	taxi-company-outline	14
CompaniaDeTaxis	TaxiCompany	clase	taxi-company-outline	14
compania-de-taxis-esquema	taxi-company-outline	proyecto	taxi-company-outline	14
crearPasajero	createPassenger	método	taxi-company-outline	14
dejarPasajero	offloadPassenger	método	taxi-company-outline	14
destino	destination	campo	taxi-company-outline	14
destinos	destinations	campo	taxi-company-outline	14
elegirUbicacionDelDestino	chooseTargetLocation	método	taxi-company-outline	14
estaLibre	isFree	método	taxi-company-outline	14
getDestino	getDestination	método	taxi-company-outline	14
getUbicacion	getLocation	método	taxi-company-outline	14
getUbicacionDelDestino	getTargetLocation	método	taxi-company-outline	14
getUbicacionDeSalida	getPickupLocation	método	taxi-company-outline	14
limpiarUbicacionDelDestino	clearTargetLocation	método	taxi-company-outline	14
Minibus	Shuttle	clase	taxi-company-outline	14
notificarArriboASalida	notifyPickupArrival	método	taxi-company-outline	14
notificarArriboDePasajero	notifyPassengerArrival	método	taxi-company-outline	14
Pasajero	Passenger	clase	taxi-company-outline	14
PasajeroFuente	PassengerSource	clase	taxi-company-outline	14
pasajeros	passengers	campo	taxi-company-outline	14
recoger	pickup	método	taxi-company-outline	14
salida	pickup	campo	taxi-company-outline	14
setUbicacion	setLocation	método	taxi-company-outline	14
setUbicacionDelDestino	setTargetLocation	método	taxi-company-outline	14
setUbicacionDeSalida	setPickupLocation	método	taxi-company-outline	14
solicitarViaje	requestPickup	método	taxi-company-outline	14
Ubicación	Location	clase	taxi-company-outline	14

Español	Inglés	Qué es	Proyecto	Capítulo
ubicacionDelDestino	targetLocation	campo	taxi-company-outline	14
Vehiculo	Vehicle	clase	taxi-company-outline	14
vehiculos	vehicles	campo	taxi-company-outline	14
fuente	source	variable local	taxi-company-outline-test	14
PasajeroFuenteTest	PassengerSourceTest	clase	taxi-company-outline-test	14
testCreacion	testCreation	método	taxi-company-outline-test	14
ubicacionDelTaxi	taxiLocation	variable local	taxi-company-outline-test	14
compania-de-taxis-etapa-uno	taxi-company-stage-one	proyecto	taxi-company-stage-one	14
destino	target	variable local	taxi-company-stage-one	14



Índice analítico

D

E

F

G

H

N

O

P

R

S

T

U