Studio Press
8 413042 751977 00011

Todo Programación

Año 1 • Número 11 • **6,50** euros

Doble CD-ROM:
Incluye Monoppix

La Revista mensual para entusiastas de la programación

www.iberprensa.com

Almacenamiento seguro de datos

Desarrollo web = PHP + MySQL

Imprimir desde Java y .NET

Nuevo curso de programación en Qt

■ DOBLE CD-ROM: MONOPPIX, UN LINUX CON MONO DE SERIE, Y COLECCIÓN DE UTILIDADES PARA EL PROGRAMADOR

Zona Linux

- **C#:** Generación y compilación de código automática
- **Actualidad:** Novedades en lenguajes y compiladores

Zona Windows

- Construye tus propias extensiones para **Firefox**
- **Reflection:** Creando ensamblados dinámicamente

NUEVO CURSO DESDE CERO: PROGRAMACIÓN DE GRIDS CON GLOBUS

Todo Programación
La Revista mensual para entusiastas de la programación

DIRECTOR
Eduardo Toribio
etoribio@iberprensa.com

REDACCIÓN
Yenifer Trabadela
yenifer@iberprensa.com

COLABORADORES
Antonio M. Zugaldía
(azugaldia@iberprensa.com)
David Santo Orcero
(orcero@iberprensa.com)
Manuel Domínguez
(mdominguez@iberprensacom)
Fernando Escudero
(fescudero@iberprensa.com)
José Manuel Navarro
(jnavarro@iberprensa.com)
Marcos Prieto
(mprieto@iberprensa.com)
Guillermo "el Guille" Som
(elguille@iberprensa.com)
Santiago Márquez
(smarquez@iberprensa.com)
José Rivera
(jrivera@iberprensa.com)
Jaime Anguiano
(janguiano@iberprensa.com)
Alejandro Serrano
(aserrano@iberprensa.com)
Jordi Massaguer
(jmassague@iberprensa.com)

DISEÑO PORTADA
Antonio Gª Tomé

MAQUETACIÓN
Antonio Gª Tomé

DIRECTOR DE PRODUCCIÓN
Carlos Peropadre
cperopadre@iberpreresa.com

ADMINISTRACIÓN
Marisa Cogorro

SUSCRIPCIONES
Tel: 91 628 02 03
suscripciones@iberprensa.com

FILMACIÓN: Fotpreim Duvial
IMPRESIÓN: I. G. Printone
DUPLICACIÓN CD-ROM: M.P.O.

DISTRIBUCIÓN
S.G.E.L.
Avda. Valdelaparra 29 (Pol. Ind.)
28108 Alcobendas (Madrid)
Tel.: 91 657 69 00

EDITA: Studio Press
www.iberprensa.com

REDACCIÓN, PUBLICIDAD Y ADMINISTRACIÓN
C/ del Río Ter, Nave 13.
Polígono "El Nogal"
28110 Algete (Madrid)
Tel.: 91 628 02 03*
Fax: 91 628 09 35
(Añeda 34 si llama desde fuera de España.)

Todo Programación no tiene por qué estar de acuerdo con las opiniones escritas por sus colaboradores en los artículos firmados.
Los contenidos de **Todo Programación** son copropiedad de Iberprensa y sus respectivos autores.

Iberprensa es una marca registrada de **Studio Press**.

DEPÓSITO LEGAL: M-13679-2004
Número 11 • Año 1
Copyright 1/6/05
PRINTED IN SPAIN

EDITORIAL

Eduardo Toribio

Empezando con Mono

En este nuevo número de **Todo Programación** podéis ver que hemos incluido un doble CD-ROM, el primero de ellos es el habitual con los listados y ejemplos de los distintos tutoriales y prácticas de la revista. El segundo CD-ROM lo hemos adjuntado por "petición popular", ya que algunos lectores nos venían solicitando una forma sencilla para empezar a trabajar con la plataforma Mono. Bien, pues con Monoppix la petición ha sido concedida, ya que se trata de un sistema Linux "live" basado en Knoppix y por tanto en Debian, donde se ofrece listas para funcionar las distintas herramientas que configuran la plataforma Mono, además de una colección de tutoriales y de ejemplos prácticos para empezar a programar en dicho entorno. Todo sin necesidad de instalar un nuevo sistema en el equipo, si bien Monoppix si ofrece esa posibilidad si así lo deseamos.

Pero como no solo de Mono vive el programador, en este número comenzamos dos nuevos cursos que creo os serán de gran utilidad debido a la importancia que están adquiriendo las herramientas que abordamos: el primer curso es sobre Globus y la programación de sistemas Grid. El segundo está dedicado a la librería de desarrollo multiplataforma Qt, entorno en el que, por ejemplo, se basa el escritorio de Linux KDE. Son dos temas muy demandados y que, por tanto, entendemos deben tener ya un hueco en TP.

SUSCRIPCIONES

Como oferta de lanzamiento existe la posibilidad de suscribirse durante 12 números a **Todo Programación** por solo 61 euros lo que significa un ahorro del 20% respecto el precio de portada. Además de regalo puedes elegir entre una de las dos guías que aparecen abajo.
Más información en: www.iberprensa.com

SERVICIO TÉCNICO

Todo Programación dispone de una dirección de correo electrónica y un número de Fax para formular preguntas relativas al funcionamiento del CD-ROM de la revista.
e-mail: todoprogramacion@iberprensa.com
Fax: 91 628 09 35

LECTORES

Comparte con nosotros tu opinión sobre la revista, envíanos tus comentarios, sugerencias, ideas o críticas.

Studio Press
(Todo Programación)
C/ Del Río Ter, Nave 13
Pol. "El Nogal"
28110 Algete. Madrid

DEPARTAMENTO DE PUBLICIDAD

Si le interesa conocer nuestras tarifas de publicidad no dude en ponerse en contacto con nuestro departamento comercial:

- **Tel. 91 628 02 03**
- **e-mail: publicidad@iberprensa.com**

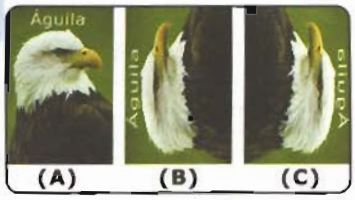
Sumario

Número 11

En portada

Imprimir desde Java y .Net

10 La mayoría de aplicaciones están centradas en el procesamiento de datos. Lejos todavía de la "oficina sin papel" todos estos datos acaban saliendo por impresora de una u otra forma, así que necesitamos algunas rutinas al respecto. Vamos a dedicar el tema de portada de este número a la impresión, nos centraremos en .NET y Java y veremos cómo ninguna de estas dos plataformas nos libran de ciertos procesos de preparación de la presentación, pero ambas nos ocultan las interioridades del hardware.



A quién vamos dirigidos

Todo Programación (TP) es una revista para programadores escrita por programadores y con un enfoque eminentemente práctico. Trataremos de ser útiles al programador, tanto al profesional como al estudiante. Si hay algo cierto en este sector es que nunca podemos parar, vivimos en un continuo proceso de reciclaje. Ahí es donde tratará de encajarse TP: información, actualidad, cursos y prácticas de los lenguajes más demandados y formación en sistemas.

ZONA LINUX

Actualidad: Novedades en lenguajes y compiladores >>

26 Iniciamos un recorrido por la actualidad de los compiladores y algunos lenguajes con el entorno .NET, y por tanto con Mono. En concreto, nos centraremos en la nueva versión 4 del compilador GCC y algunas características de C# 2.0.

C#: Generación y compilación de código automática >>

29 Estudiamos cómo crear aplicaciones de forma abstracta, generar sus fuentes en diferentes lenguajes de programación, así como compilarlas y ejecutarlas.

Curso de programación con GTK# >>

32 Vamos a realizar un ejemplo práctico para continuar aprendiendo a trabajar con GTK#, en concreto desarrollaremos una sencilla aplicación para GNOME utilizando el lenguaje C# y la librería GTK#.

CONTENIDO DEL CD-ROM 1

CD 1: Utilidades para el programador

63 En el CD-ROM se pueden encontrar las utilidades y herramientas de programación más útiles, además de todas las aplicaciones y ejemplos que sean necesarios para poder seguir los tutoriales y cursos de la revista. En este número incluimos las versiones de Qt para Windows y Linux, además de un entorno de desarrollo MDA, diversas aplicaciones de mirroring para Windows y la distribución estándar de Perl.

CONTENIDO DEL CD-ROM 2

CD 2: Monoppix

66 Seguramente Knoppix es un término ya conocido por todos, un sistema operativo Linux basado en la popular Debian e instalable en modo vivo, o lo que se llama una distribución live. Bien, pués Monoppix es un sistema Knoppix modificado para incluir todo lo necesario para ejecutar Mono, además de numerosos ejemplos prácticos y una amplia colección de tutoriales.

TALLER PRÁCTICO

Construye tus propias extensiones para Firefox >>

39 Vamos a estudiar cómo trabajar con el formato XUL, un lenguaje de marcas mediante el cual es posible programar extensiones para el navegador libre Firefox, que cada día gana más terreno a Microsoft Internet Explorer.

Introducción al framework de Qt: El ahorcado >>

44 Comenzamos un minicurso de programación en Qt, un conjunto de utilidades que hacen que para crear el mismo interfaz para los sistemas Windows, Linux, AIX, HP-UX o Solaris solo sea necesario recompilar. Comenzaremos además con un ejemplo práctico.

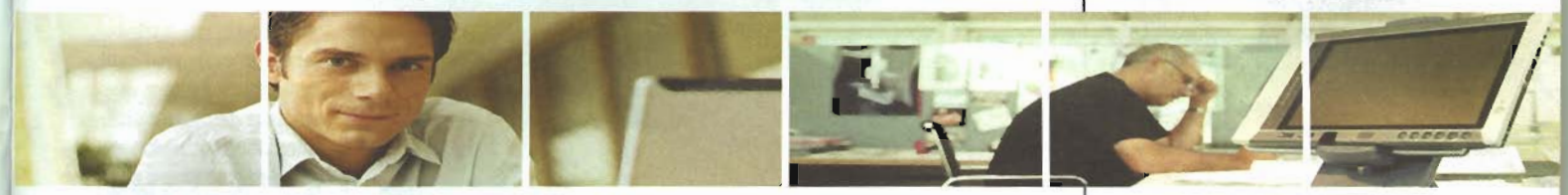
Programación de Grids: Globus >>

47 Iniciamos una miniserie dedicada a la programación de sistemas grid. En concreto, nos centraremos en Globus, un toolkit multiplataforma que permite desarrollar aplicaciones que funcionen con semántica de grid.

Y ADEMÁS...

Desarrollo Web: MySQL en PHP >>

52 La combinación del lenguaje PHP con un motor de bases de datos utilizando MySQL es la base de un gran número de weblogs que podemos visitar a día de hoy en Internet. Dedicamos esta entrega de la serie a estudiar dicha combinación.



.net ZONA WINDOWS

Reflection: Creando ensamblados dinámicamente >>

35 Segunda parte de nuestra miniserie dedicada a Reflection. Si en la primera vimos cómo obtener toda la información almacenada en los ensamblados, ahora nos centraremos en la creación de éstos en tiempo de ejecución.

Reportaje: Almacenamiento seguro >>

21 El incendio en el rascacielos Windsor de Madrid ha preocupado a muchos responsables de departamentos de informática en todo el país. ¿Estarán seguros mis datos? La realidad es que muchas veces la respuesta es "no". Dedicamos estas páginas a ver qué podemos hacer para tener siempre a salvo nuestros datos.

Juegos Java: Comunicaciones >>

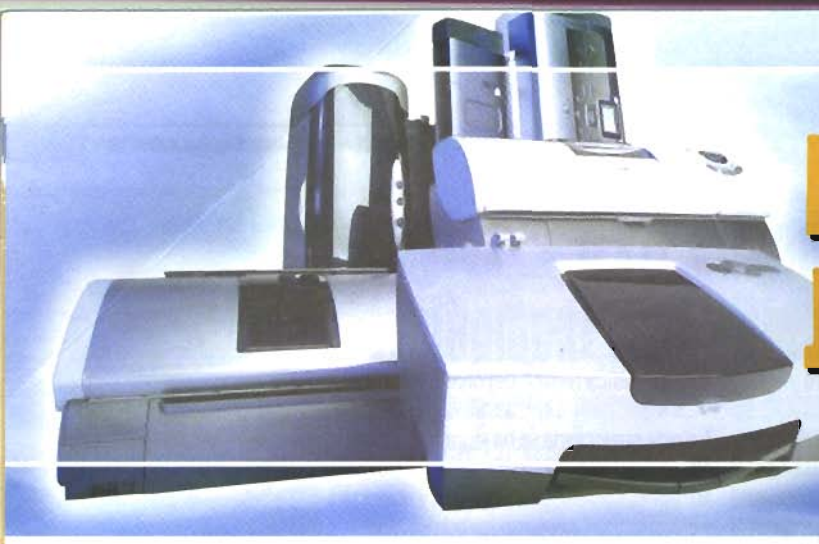
55 Nos adentramos en las posibilidades que nos proporciona el API para realizar tareas de comunicación, veremos las ventajas y facilidades que nos ofrece, así como los inconvenientes o aspectos que todavía no están lo suficientemente maduros como para considerarlos en nuestros desarrollos.

Bases de datos: Vistas y consultas anidadas en PL/SQL >>

59 Llegamos al final de nuestra serie dedicada a SQL y las Bases de Datos. En este último artículo vamos a ver aquellos conceptos que han quedado sin tratar en las entregas anteriores: las vistas, y las consultas anidadas.

NOTICIAS

- **6. IBM** en CeBIT 2005.
- **6. Sun** Ray Server amplia sus funcionalidades.
- **7. Acuerdo** entre Microsoft y Telefónica móviles.
- **7. VERITAS** ofrece soporte para Solaris 10.
- **8. Premios** Development Jolt.
- **8. Cyclades** lanza AfterPath BladeManager.
- **8. Acuerdo** entre Sun y la Universitat Autónoma de Barcelona.
- **8. Boland** Core SDP.
- **9. Plataforma** portátil Quosmio G20.
- **9. Workstation** HP xw9300.
- **9. Nueva** gama de portátiles Acer Aspire.



Imprimir desde Java y .NET

Manuel Domínguez (Java) mdominguez@iberprensa.com
Guillermo Soms "el Guille" (.NET) elguille@iberprensa.com

La mayoría de las aplicaciones actuales están centradas en el procesamiento de datos, de una u otra forma. Y lejos aún de la "oficina sin papel", todos estos datos acaban saliendo por una impresora tarde o temprano. Así que necesitaremos algunas rutinas al respecto. Pero imprimir desde .NET y Java no es ni fácil ni complicado, simplemente hay que saber cómo hacerlo correctamente.

Todas las aplicaciones actuales están basadas en datos de tipo gráfico, texto, números, etc. En realidad, la informática se inventó para poder tratar datos de forma automatizada con una máquina. Se introducen por un extremo, la máquina hace algo con ellos según el programa que utilicemos y nos los devuelve procesados, generalmente en papel.

Para los desarrolladores de cualquier plataforma eso significa que no solo han de poner mucha atención a las rutinas de entrada de datos y las de procesamiento, sino que, tarde o temprano, han de enfrentarse a las peticiones de sus usuarios, que quieren llevarse los resultados obtenidos en una carpeta a la reunión de turno. Y eso significa "pelearse" con las impresoras y las rutinas correspondientes en los programas. Tanto Java como .NET no lo ponen difícil pero, sin duda, hay que saber cómo hacerlo correctamente, si no queremos terminar con un ataque de frustración. El remedio para este poco halagüeño escenario lo explicamos a continuación.

Imprimir desde Java con AWT

Java cuenta con dos paquetes para generar salidas por la impresora: *java.awt.print* y *javax.print*. El primero de ellos es el más básico y el que comentaremos en este apartado. El segundo es más complejo y para entenderlo sería bueno estar familiarizado con el primero. Dejamos pues en manos del lector aventurarse en *javax.print* tras la lectura de esta sección.

SENTAR LAS BASES

Antes de lanzarnos a tumba abierta a imprimir los resultados del procesamiento de nuestras aplicaciones, hay que tener en cuenta algunos detalles particulares de la plataforma de la taza. Pocos, eso sí, pero importantes.

Maquetado de página

Como en el resto de lenguajes de programación, en Java hay que crear en memoria la página que se desea enviar a la impresora, de tal forma que una vez maquetada dicha página, solamente tenemos que seleccionar la impresora e imprimir. En esta operación entran en juego diversas clases, métodos e interfaces Java que iremos explicando. Para todos los ejemplos de este reportaje se deberá importar el paquete *java.awt.print*.

Interfaces

En AWT la forma de imprimir es relativamente sencilla. Cada objeto a imprimir debe implementar un interfaz que permitirá la impresión. Esta operación consiste, básicamente, en implementar en el objeto a imprimir un método que será el que formará las páginas, y las dejará construidas para que el proceso encargado de enviarlas a la impresora las tome y las imprima sin más. Por ejemplo, supongamos que tenemos un objeto que es una factura; a la hora de imprimir una factura uno tiende a pensar que el programa principal es quien toma los datos de la misma y genera las páginas de impresión. En realidad no es así, es el propio objeto *factura* el que tiene un método interno que crea la imagen de la factura en memoria y la devuelve cuando el motor de impresión se lo solicita. Para ello el API de Java proporciona dos interfaces importantes: *Printable* y *Pageable*, que veremos a continuación.

El interfaz Printable

Printable es el interfaz utilizado por objetos cuya salida impresa va a consistir en un conjunto de páginas con las mismas características. Este tipo de documento suele ser lo más habitual, y sus páginas, todas con el mismo formato, están numeradas consecutivamente a partir de la 1.

La implementación de *Printable* implica, como ya dijimos, la implementación del método *print*. Éste acepta como parámetros un objeto *Graphics*, un objeto *PageFormat* y un entero. Estos parámetros sirven para decir al objeto: "Dibújame

Figura 1. Esquema general.

sobre este objeto *Graphics* la página X (1, 2, etc.) con el formato de página tal (A4, B5, personalizado...)". Asimismo *print* devuelve un entero que indicará si se ha maquetado una página *(constante PAGE_EXISTS)* o si no existen más páginas que maquetar *(constante NO_SUCH_PAGE)*. En definitiva, *print* es el método encargado de maquetar páginas cuando se le solicite, y éstas serán enviadas automáticamente a la impresora. Cuando no tiene más páginas que formar, devuelve un valor para indicárselo al motor de impresión que finaliza la operación.

En el objeto *Graphics* que obtiene por parámetro el método *print*, se dibuja, usando el API proporcionado por *Graphics*, todo lo que se desea imprimir en papel. En números anteriores de **Todo Programación** ya realizamos una introducción a las clases y métodos para dibujar en Java. Se usarán exactamente igual, con la salvedad de que lo que antes se dibujaba con idea de ser mostrado en la pantalla, ahora se dibuja y se prepara para enviarse a la impresora. Por lo demás, si tenéis a mano el número 8 de la revista donde se explica la clase *Graphics*, éste puede ser de mucha ayuda.

En ningún caso se llamará al método *print* de un objeto *Printable* directamente, sino que será el motor de impresión el que lo hará automáticamente repetidas veces, indicándole en cada llamada el número de página que debe construir y devolver, hasta que no quede ninguna por imprimir.

Generalmente en un objeto que implemente el interfaz *Printable*, su método *print* será sencillo de implementar cuando se trate de generar un número fijo o reducido de páginas o bien de poca complejidad. Por el contrario, se complicará cuando el número de páginas crezca o éstas sean más laboriosas. Una posible implementación del interfaz *Printable* puede verse en el ejemplo *MiTipoPrintable1.java* incluido en el CD de la revista o bien en el Listado 1.

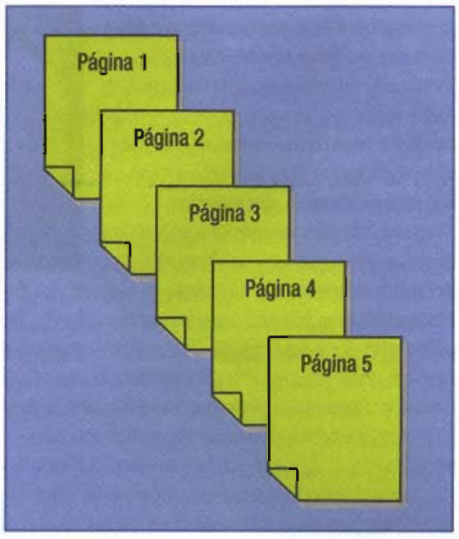


Figura 2. Aspecto de un documento printable.

En este caso, se crea una clase que implementa el interfaz *Printable*. El método *print* devuelve la hoja número 0 con un rectángulo que la ocupa entera, y la hoja numerada con 1, con un óvalo que la ocupa también en su totalidad. No devuelve más páginas. Éste es un ejemplo muy básico.

Para la impresión de informes asociados a datos de una base de datos, que suele ser muy usual, el método *print* tendría otro aspecto muy distinto. En esas ocasiones se suelen utilizar librerías que, a un nivel mucho más abstracto, facilitan esta labor, como por ejemplo *JfreeReport*.

El interfaz Pageable

El método de impresión anterior es bueno, pero carece de algunas características importantes. La más destacada es el hecho de que con el interfaz *Printable* todas las páginas deben ser iguales en cuanto a su formato. Si bien la mayoría de los documentos son así, ciertamente hay muchos otros casos en los que esto no ocurre y las páginas que difieren en cuanto a tamaño, orientación, márgenes, márgenes para impresión, etcétera. Este tipo de documentos, más complejos, deben implementar otro interfaz proporcionado por el API de Java: el interfaz *Pageable* o, como ya veremos más adelante, hacer uso o heredar de clases que ya la implementen.

Este interfaz obliga a las clases que la implementen a tener tres métodos: *getNumberOfPages()*, cuyo cometido es devolver el número de páginas que contiene el objeto *Pageable*; *getPageFormat (int numPag)* para devolver el formato de la página especificada por parámetro, y por último el método *getPrintable(int numPag)* que devuelve el objeto *Printable*, encargado de maquetar la página especificada por parámetro.

En el CD de la revista se encuentra el ejemplo *MiTipoPageable1.java* que implementa el interfaz *Pageable* y cuyo código se puede observar en el Listado 2 de la página siguiente. Algunas líneas de este ejemplo no se entenderán bien hasta que no se lean los siguientes párrafos, pero valdrá como introducción.

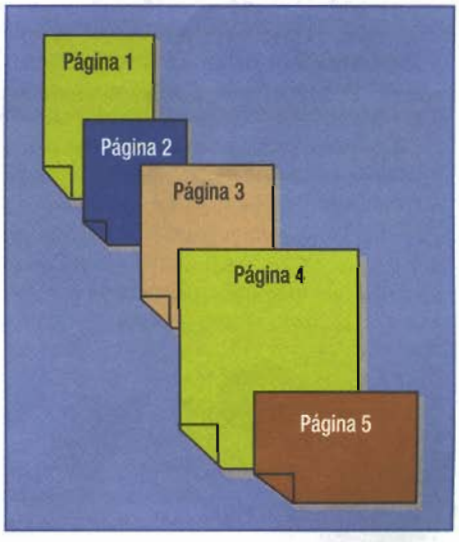


Figura 3. Aspecto de un documento pageable.

En definitiva, cualquier objeto *Pageable* se puede entender como una colección de pares donde cada par consta de un objeto *Printable*, que general-

Listado 1

```
public int print(java.awt.Graphics graphics, PageFormat pageFormat, int
pageIndex)
                    throws PrinterException {
        double x = pageFormat.getImageableX();
        double  y = pageFormat.getImageableY();
        double ancho = pageFormat.getImageableWidth();
        double alto = pageFormat.getImageableHeight();
        if (pageIndex == 0) {
            graphics.drawRect((int) x, (int) y, (int) ancho, (int) alto);
            return PAGE_EXISTS;
        } else if (pageIndex == 1) {
            graphics.drawOval((int) x, (int) y, (int) ancho, (int) alto);
            return PAGE_EXISTS;
        } else
            return NO_SUCH_PAGE;
    }
```

mente imprimirá una sola página, y de un *PageFormat*, que más adelante veremos cómo funciona. Es decir, una colección de páginas cada una con un formato concreto y donde el acceso a ellas se realiza de forma directa por el índice o número de página. Como vemos, el interfaz *Pageable* va un paso más allá, permitiendo documentos mucho más versátiles que *Printable*.

LAS CLASES PARA IMPRIMIR CON AWT

Además de los interfaces vistos en los puntos anteriores, el paquete *java.awt.print* incorpora algunas clases, unas se deben utilizar obligatoriamente para el proceso de impresión, mientras que el uso de otras es optativo pero existen para facilitar el trabajo o acelerarlo. En las siguientes líneas veremos las más importantes e iremos completando todas las piezas necesarias para imprimir desde Java.

La clase Book

La clase *Book* es una implementación del interfaz *Pageable*. Java aporta esta implementación, ya que con ella se suelen dar respuesta a cualquier tipo de documento a imprimir. Por tanto, *Book* implementa los métodos del interfaz *Pageable*, aunque además incorpora un buen número de métodos que permitirán trabajar con documentos complejos de una forma sencilla.

El método más interesante de la clase *Book* es *append(Printable p, PageFormat pf)* que tiene como cometido añadir una nueva página al conjunto de páginas (al documento). Para cada página obliga a especificar un objeto *Printable*, que se encargará de maquetarla, y un objeto *PageFormat* que indica el formato que tendrá la página. Así vemos como en el caso del interfaz *Pageable* y, concretamente, en la clase *Book* que lo implementa, acaba conteniendo objetos *Printable* y un formato de página asociado a cada uno de ellos. Por norma, siempre es mejor utilizar objetos *Pageable* en lugar de *Printable* porque proporcionan mayor flexibilidad y potencia.

En el **Listado 2** se encuentra el uso de *Book* en un objeto de tipo *Pageable*. *Book* es *Pageable*, así que la clase *MiTipoPageable1* que hereda de ella, también lo es.

Hasta ahora hemos aprendido qué son objetos *Printable* y sabemos que son imprimibles por sí mismos. Además, hemos averiguado qué son objetos *Pageable* (colecciones de objetos *Printable*). Los objetos *Printable* también son imprimibles directamente por el motor de impresión, al igual que el objeto *Pageable*. Y la impresión es asimismo automática porque el motor de impresión se encarga de invocar para cada una de las páginas a su "maquetador", para generar una página con el formato apropiado.

La clase Pageformat

La clase *PageFormat* se utiliza para indicar el formato de papel que debe tenerse en cuenta a la hora de imprimir una página. Es uno de los parámetros del método *print* de un objeto *Printable*, por tanto, siempre se debe tener en cuenta a la hora de implementar el objeto *print*. De lo contrario, nos podemos encontrar con la sorpresa de que lo que deseamos imprimir no queda centrado, o incluso a lo mejor ni cabe en el papel, una vez sale de la impresora.

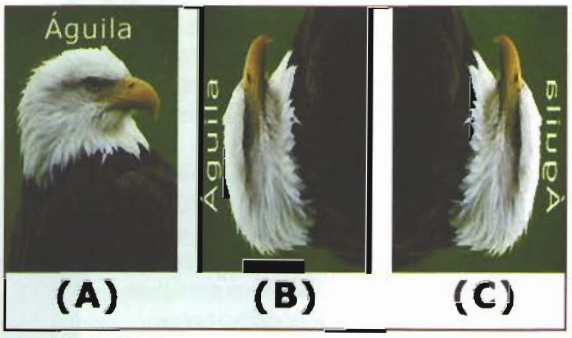


Figura 4. (A) Portrait, (B) Landscape y (C) Reverse landscape.

Esta clase especifica y permite obtener principalmente la orientación del papel, así como el ancho y alto del área de impresión y el origen de la misma. El método *setOrientation(int o)* se puede invocar pasando por parámetro una de las siguientes constantes: *LANDSCAPE*, *PORTRAIT* y *REVERSE_LANDSCAPE*, que indicarán que la página a imprimir tendrá formato vertical, apaisada o vertical-reflejada, respectivamente. Asimismo este valor se puede obtener con *getOrientation()*. Otros métodos importantes son *getImageableX()* y *getImageableY()* que devuelven la coordenada X e Y del punto más a la izquierda y arriba que se puede imprimir en el papel con el formato elegido, y por último, *getImageableWidth()* y *getImageableHeight()* que permiten obtener el ancho y alto del área de impresión.

Son especialmente importantes dos métodos que permiten definir el tipo de papel, además de la orientación del mismo: *setPaper(Paper p)* y *getPaper()*. El primero permite establecer el papel a utilizar mediante un objeto de la clase *Paper*, y el segundo obtener ese dato. En las próximas líneas veremos la clase *Paper* y sus métodos principales. Existen otros métodos, pero tienen mucha menor utilidad.

La clase Paper

La clase *Paper* identifica el tamaño del papel que se va a usar en la impresión. Es el modo de elegir, por ejemplo, si trabajar con A4, con Letter, con B5, etcétera. Dispone de cuatro métodos importantes:

Listado 2

```
public class MiTipoPageable1 extends Book {
    /** Este objeto representa un documento con dos páginas. Una vertical
     * y la otra apaisada que contienen lo mismo.
     */
    public MiTipoPageable1() {
        PageFormat formatoPagina0 = new PageFormat();
        PageFormat formatoPagina1 = new PageFormat();
        Paper papelPagina0 = new Paper();
        papelPagina0.setSize(192.4251, 283.4645);
        papelPagina0.setImageableArea(0, 0, 200, 300);
        Paper papelPagina1 = new Paper();
        papelPagina1.setImageableArea(0, 0, 200, 300);
        formatoPagina0.setPaper(papelPagina0);
        formatoPagina1.setPaper(papelPagina1);
        formatoPagina0.setOrientation(PageFormat.PORTRAIT);
        formatoPagina1.setOrientation(PageFormat.LANDSCAPE);
        this.append(new MiTipoPrintable2(), formatoPagina0);
        this.append(new MiTipoPrintable2(), formatoPagina1);
    }
}
```

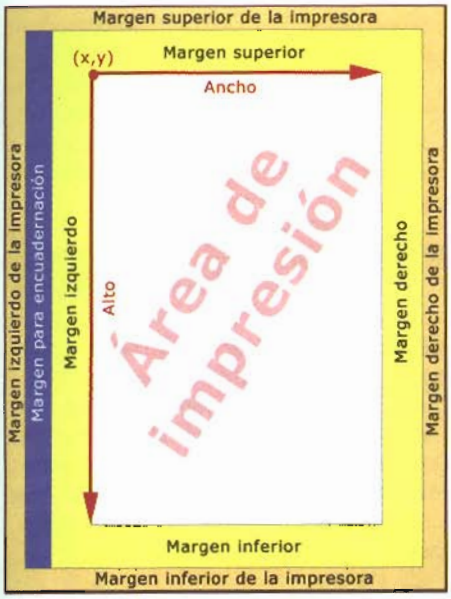


Figura 5. Partes típicas de una página impresa.

getImageableX(), *getImageableY()*, *getImageableWidth()* y *getImageableHeight()*, que tienen exactamente la misma finalidad que sus homólogos en la clase *PageFormat*. Además, tiene otros métodos que la convierten en una clase muy útil (de hecho, es imprescindible), como por ejemplo *setImageableArea (double X, double Y, double W, double H)*, que permite establecer el área de impresión de la página especificando el punto más a la izquierda y arriba que se imprimirá, además de un ancho y largo a partir de él para delimitar el rectángulo.

Pero, sin duda, el método más importante de esta clase es *setSize(double W, double H)* que permite especificar el ancho y el alto del papel, es decir, el tamaño del mismo. El ancho y el alto se debe medir en unidades de 1/72 de pulgada, ya que es así como lo especifica el API. Esto es necesario porque la conversión entre resolución de pantalla y resolución o medida de impresión ha de hacerse en este punto. A nadie se le escapa que una imagen mostrada en el monitor tiene otra medida generalmente cuando se imprime. Una pulgada equivale a 2,54 centímetros, así que para convertir los centímetros a unidades de impresión habría que seguir la formula siguiente:

```
UnidadesdeImpresion =
(Centímetros / 2'54) * 72
```

O, lo que es lo mismo, hay que multiplicar el valor en centímetros por 28'34645669 y el resultado estará en las unidades esperadas por el método *setSize(double W, double, H)* de la clase *Paper*.

Por ejemplo, para especificar un papel de tamaño 7x10 centímetros tendríamos que pasar ambos valores a unidades de impresión. 7 se convertiría en (7/2'54)*72= 192'4251 y 10 en (10/2'54)*72=283'4645. Y, por tanto, invocaríamos al método como *miPapel.setSize(192'4251, 283'4645)*. Realmente es un proceso tedioso, pero veremos que en realidad muy rara vez utilizaremos esta función directamente, sino que seleccionaremos el tamaño de la página a través de un cuadro de diálogo. En cualquier caso, recomendamos implementar un método que realice esta operación, y usarlo directamente. Si no, ocurre que al leer el código no se está seguro de los centímetros que se han especificado.

La clase Printerjob

PrinterJob es lo que durante todo el artículo hemos llamado abstractamente "motor de impresión". Siempre crearemos una instancia de esta clase en nuestra aplicación, y ella será la encargada en todo momento de imprimir el documento. Para ello hay que indicarle qué documento es el que deseamos imprimir y, eventualmente, mostrar el cuadro de diálogo de configuración de la impresora o la página. *PrinterJob* es una clase abstracta, lo que significa que no se puede instanciar; en su lugar invocaremos directamente a un método de la clase para crear nuestro motor de impresión, como se muestra a continuación:

```
PrinterJob miMotor =
PrinterJob.getPrinterJob();
```

Y a partir de ahí es donde comienza realmente a existir en nuestra aplicación un módulo encargado de imprimir. Por

Listado 3

```
public class UsandoPrinterJob1 {
    /** Creates a new instance of prueba */
    public UsandoPrinterJob1() {
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        PrinterJob motor = PrinterJob.getPrinterJob();
        motor.setPageable(new MiTipoPageable1());
        try {
            motor.print();
        } catch (Exception e) {}
    }
}
```

Listado 4

```
public class UsandoPrinterJob2 {
    /** Creates a new instance of prueba */
    public UsandoPrinterJob2() {
    }
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        PrinterJob motor = PrinterJob.getPrinterJob();
        motor.setPageable(new MiTipoPageable1());
        try {
            if (motor.printDialog()) {
                motor.print();
            }
        } catch (Exception e) {}
    }
}
```

supuesto, a partir de ahora *miMotor* nos ofrecerá un conjunto de métodos que cualquier objeto *PrinterJob* tiene. Dos de ellos, *setPrintable(Printable p)* y *setPageable(Pageable p)* sirven para indicar al motor de impresión cuál es el documento que deseamos imprimir. El primero permite especificar un documento de tipo *Printable*, como los que hemos visto anteriormente, mientras que si queremos imprimir un objeto de tipo *Pageable*, deberemos usar el segundo de los dos métodos. También contamos con *printDialog()*, que mostrará en pantalla un diálogo de impresión para este motor de impresión, en el que podremos seleccionar la impresora y sus propiedades, así como el tipo de papel y su formato. Por último, cuando tengamos el documento a imprimir y las características de la impresión, llamaremos al método *print()* del *PrinterJob* que es el que finalmente inicia la impresión del documento en la impresora física.

En el **Listado 3** y **Listado 4** de la página anterior se pueden observar dos ejemplos. Estos ejemplos completos se encuentran en el CD de la revista, con el nombre *UsandoPrinterJob*. En el primero de ellos se hace uso de un *PrinterJob* para imprimir un objeto *Pageable* de tipo *MiTipoPageable1*. En el segundo listado, se realiza la misma operación pero mostrando primero un cuadro de diálogo que permite la selección de la impresora, tamaño de papel, etcétera.

PARA TERMINAR: EXCEPCIONES

Como en casi cualquier paquete Java, y sobre todo en aquellos que hacen uso de la entrada/salida, el paquete *java.awt.print* incorpora algunas excepciones que nos permitirán manejar correctamente situaciones especiales durante el proceso de impresión desde nuestra aplicación. Concretamente una aplicación Java puede lanzar tres excepciones con respecto al proceso de impresión (Ver **Listado 4**):

PrinterAbortException, que será lanzada por Java cuando el usuario o la aplicación hayan abortado una impresión que estaba en curso. Generalmente ocurre cuando se elimina el trabajo de impresión de la cola de impresión del sistema operativo.

PrinterIOException ocurre cuando se produce un error de entrada/salida en relación con la impresora, como por ejemplo apagar la impresora mientras estaba imprimiendo. Esta excepción cuenta con un método *getCause()* que permite obtener la causa del salto de la excepción.

Por último, una excepción genérica llamada *PrinterException* que se lanzará cuando haya ocurrido una excepción no cubierta por alguna de las dos situaciones descritas anteriormente.

CONCLUSIÓN

Aunque existe un API de impresión más avanzado en Java, *javax.print*, en la mayoría de las ocasiones en las que nos surge tener que imprimir algo, es más que suficiente con lo que hemos visto. Resulta muy sencillo, y sin duda dotará a nuestras aplicaciones de una funcionalidad extra. Además, ya que la API de Java incorpora estas clases... ¿vamos a dejar de usarlas?

Imprimir desde .NET

Muchos de los usuarios de Visual Basic 6 que han migrado a .NET han sufrido (incluso están sufriendo) la carencia de este "nuevo" entorno de programación de un objeto que les permita imprimir de forma tan fácil como lo era el objeto *Printer*, o de una colección para conocer las impresoras disponibles en el sistema. En esta segunda parte de nuestro tema de portada explicaremos cómo imprimir en .NET y comprobaremos que, aunque parezca complicado, a la larga es igual o más fácil que con VB6, aunque tratándose de un lenguaje orientado a objetos debemos cambiar un poco el chip

SENTAR LAS BASES

Aunque .NET pregona una alta productividad y gran facilidad de uso, es necesario saber algunos fundamentos para poder aprovecharlo, si no queremos perder mucho tiempo a la hora de escribir código para impresión. A continuación veremos los "secretos" a tener en cuenta.

Los precedentes

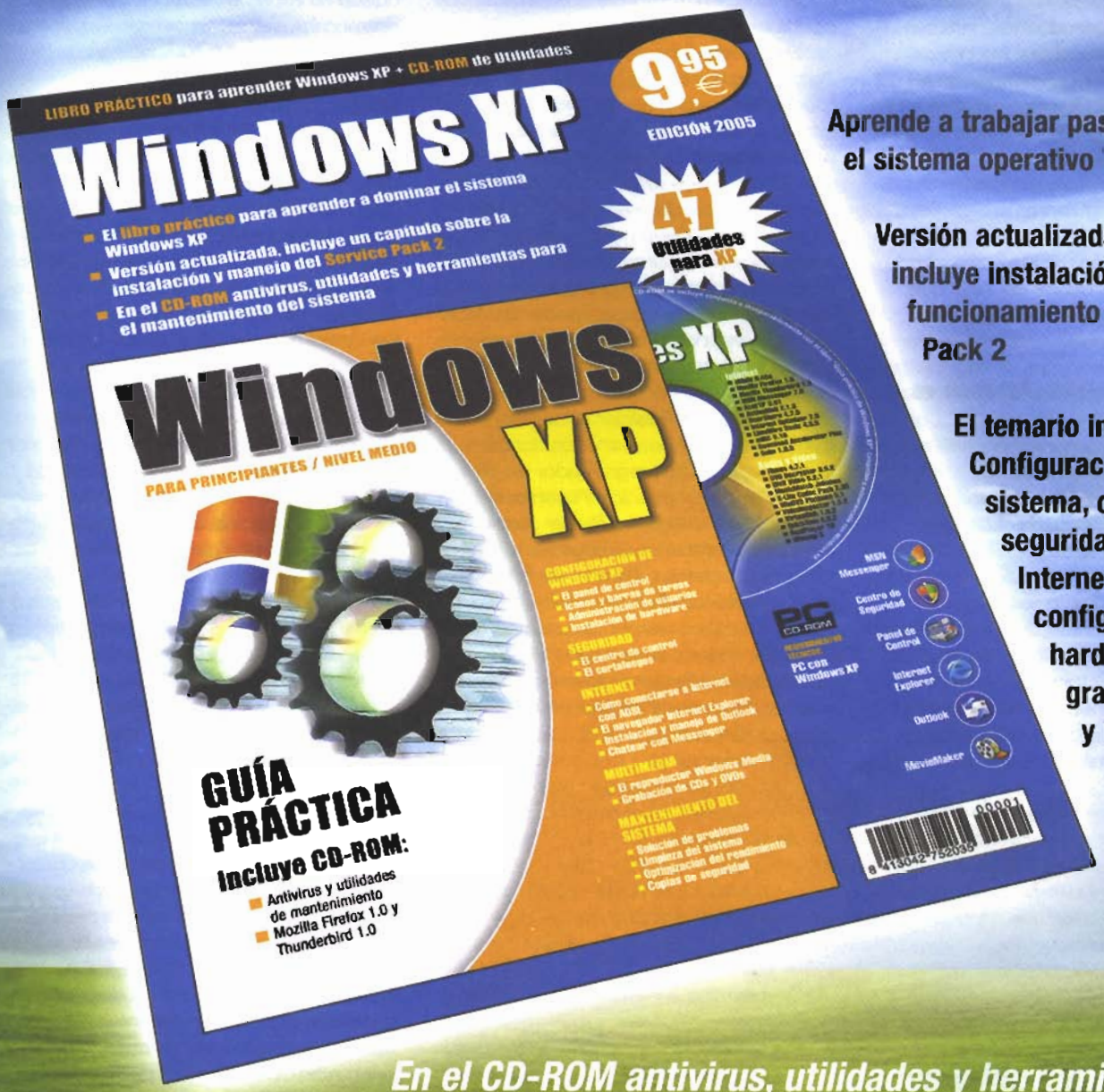
Lo primero que debemos tener en cuenta es que todo lo que queramos hacer en .NET debemos realizarlo usando las clases que este entorno nos ofrece. Y el tema de la impresión no es una excepción. En Visual Basic 6 solamente disponíamos del objeto *Printer*, que era el que realmente nos permitía imprimir, y de la colección *Printers* que podíamos utilizar para saber las impresoras que teníamos disponibles. Además estaba el control para diálogos comunes, que nos permitía seleccionar y configurar la impresora a utilizar, indicaba el número de páginas a imprimir, etc

No obstante, no siempre funcionaba como a nosotros nos hubiera gustado y en ocasiones teníamos que acudir a las funciones del API de Windows si realmen-

Cuadro 1. Java Print Service

El Servicio de Impresión de Java es un nuevo API, definido en el paquete *javax.print*, que permite la impresión en todas las plataformas en las que Java está disponible, incluso en aquellas con pocas posibilidades como, por ejemplo, a las que está destinada la plataforma J2ME. Mantiene compatibilidad hacia atrás de forma que el tradicional API de impresión, el que se ha comentado en este artículo, es perfectamente válido. El funcionamiento del nuevo Servicio de Impresión de Java se basa en la definición de atributos, siendo compatible con la formalización que el IETF (Internet Engineering Task Force) ha realizado del Protocolo de Impresión de Internet (IPP) en su versión 1.1. En este modelo, las aplicaciones pueden detectar servicios de impresión que cumplan con los atributos definidos, incluidos los servicios de impresión de flujos de datos cuyo cometido es la transformación de un flujo de datos de un formato a otro diferente.

El libro práctico para aprender a dominar el sistema XP



Aprende a trabajar paso a paso con el sistema operativo Windows XP

Versión actualizada 2005, incluye instalación y funcionamiento del Service Pack 2

El temario incluye Configuración del sistema, opciones de seguridad, conexión a Internet, configuración de hardware, grabación de Cds y DVDs, etc.

Nivel de usuario
principiante intermedio avanzado

En el CD-ROM antivirus, utilidades y herramientas para el mantenimiento del sistema. El mejor software del momento para sacar todo el provecho al PC.

Cómo recibir el libro práctico **Windows XP** en tu domicilio:

✓ Por Teléfono (de 9 a 14h y de 15 a 18 h): **916280203**

✓ Por Fax: **916280935**

✓ Por correo electrónico: *suscripciones@iberprensa.com*

Ya a la venta en quioscos y centros comerciales

te queríamos hacer algo verdaderamente práctico.

En .NET ha cambiado un poco la forma de hacer todo lo que hacíamos en VB6, ya no existen objetos que nos permitan imprimir, ni colecciones para saber las impresoras disponibles, bueno, un poco sí, pero como veremos de una forma más ordenada y algo diferente. Esa diferencia es la que a muchos les ha causado desesperación e incluso les ha llevado a un consumo excesivo de Aspirinas. En los siguientes puntos veremos que nuestra vida puede ser más soportable e incluso más fácil si aprendemos a manejar las clases de .NET que nos permiten imprimir y configurar las opciones de impresión.

La solución .NET o cómo imprimir en .NET

Básicamente, para imprimir en .NET solamente necesitamos una clase. *PrintDocument*, que está definida en el espacio de nombres *System.Drawing.Printing*. Con esta clase tenemos todo lo necesario para imprimir cualquier tipo de documento (en la impresora predeterminada). Para ello simplemente necesitamos llamar al método *Print* y asunto arreglado. Seguramente el lector se preguntará que si esto es así, ¿dónde está el problema? Problema, lo que se dice problema, realmente no existe ninguno, lo que ocurre es que esta clase, particularmente el método *Print*, se utiliza sin tener que indicar qué es lo que queremos imprimir, y aquí es donde podría existir alguna dificultad.

El método *Print* de la clase *PrintDocument* lo que realmente hace es "despertar a la bestia", es decir, dar las instrucciones pertinentes al motor de .NET para que se inicie el proceso de impresión. Dicho proceso se lleva a cabo básicamente utilizando el evento *PrintPage* de la clase *PrintDocument*, y es en este evento donde tendremos que hacer todo lo necesario para que se imprima lo que queramos. Por tanto, para poder controlar lo que se va a imprimir, deberemos escribir todo nuestro código en ese evento, que se produce para cada página que deba imprimirse. Posiblemente éste sea el punto problemático, al menos desde el punto de vista del programador de VB6, ya que antes simplemente usábamos el método *Print* del objeto *Printer* para controlar lo que debía imprimirse (o de cualquier objeto del tipo *Printer* que hubiésemos obtenido de la colección *Printers*). Le indicábamos simplemente qué es lo que debía imprimirse, aunque en VB6 éramos nosotros los que debíamos comprobar cuándo cambiar de página, cuándo imprimir la cabecera, etc. En .NET todo esto se hace en el evento *PrintPage*. Por tanto, podemos decir que en .NET es más fácil imprimir y, sobre todo, controlar cómo y dónde se imprime cada cosa, ya que, si lo simplificamos al máximo, todo se hace en un solo lugar: el evento *PrintPage* de la clase *PrintDocument*.

Para muestra, un botón

En el Listado 5 podemos ver un ejemplo de la forma mas simple de imprimir en .NET. Mostramos la cadena "Hola, Mundo" en la parte superior de la página usando una fuente Arial de 24 puntos en negrita.

Como podemos comprobar, el proceso es bien simple:

- Primero tenemos el código utilizado para dar la orden de imprimir, (este código puede estar en el evento *Click* de un botón). En este creamos un nuevo objeto del tipo *PrintDocument* al que asignamos el procedimiento de evento a usar cuando se vaya a imprimir cada página, cosa que ocurrirá después de llamar al método *Print* de esta clase.
- En el evento *PrintPage* le indicamos a .NET qué es lo que queremos imprimir, en nuestro caso solo la cadena "Hola, Mundo". Para imprimir dicho texto utilizamos el método *DrawString* del objeto *Graphics*, (una propiedad del segundo parámetro pasado al evento).

Esta sencillez se debe en parte a que tampoco realizamos demasiadas cosas en el código del evento *PrintPage*, aunque aquí mostramos algunas de las necesarias, como por ejemplo, indicar el tipo de fuente a usar para la impresión. Ésta se puede cambiar en cada una de las líneas a imprimir (e incluso en distintas partes de esa línea), ya que cada línea se "dibuja" por medio del método *DrawString* del objeto *Graphics* obtenido del segundo parámetro pasado a este método. Como vemos, en *DrawString* debemos indicar qué es lo que queremos imprimir, con qué tipo de fuente y en qué posición. Por último, señalaremos si quedan más páginas a imprimir. Esto lo haremos asignado un valor verdadero o falso a la propiedad *HasMorePages* del objeto recibido como segundo parámetro del evento. Si el valor asignado es *True*, estaremos indicando que queremos seguir imprimiendo, por el contrario, asignando un valor *False* terminaremos la impresión, algo que también ocurrirá si no le asignamos expresamente nada a esa propiedad.

Listado 5

```
' Ejemplo simple para imprimir en .NET
' (Este código estará en el evento Click de un botón)
' Usamos una clase del tipo PrintDocument
Dim printDoc As New PrintDocument
' asignamos el método de evento para cada página a imprimir
AddHandler printDoc.PrintPage, AddressOf print_PrintPage
' indicamos que queremos imprimir
printDoc.Print()

' El evento producido para cada página a imprimir
Private Sub print_PrintPage( _
    ByVal sender As Object, ByVal e As PrintPageEventArgs)
    Dim s As String = "Hola, Mundo"
    ' mostraremos la cadena en el margen izquierdo
    Dim xPos As Single = e.MarginBounds.Left
    ' La fuente a usar
    Dim prFont As New Font("Arial", 24, FontStyle.Bold)
    ' la posición superior
    Dim yPos As Single = prFont.GetHeight(e.Graphics)
    '
    ' imprimimos la cadena
    e.Graphics.DrawString(s, prFont, Brushes.Black, xPos, yPos)
    ' indicamos que ya no hay nada más que imprimir
    ' (el valor predeterminado es False)
    e.HasMorePages = False
End Sub
```

Configurar la impresión

Para ser claros, ésta no será la forma habitual de emplear este evento, ya que en la mayoría de los casos imprimiremos más de una página y seguramente querremos asignar otros valores a la impresora, como márgenes, número de copias, tamaño del papel, orientación de la impresión (vertical o apaisada) e incluso, lo más importante, qué impresora utilizar. Todos estos valores podremos indicarlos usando un objeto del tipo *PrinterSettings* que, entre otras cosas, nos permite saber cuáles son las impresoras que tenemos disponibles en el equipo actual. Por tanto, lo normal será que empleemos un objeto de esta clase en conjunto con el de la clase principal para imprimir *PrintDocument*.

Seleccionar la impresora a utilizar

Otra de las tareas que seguramente deberemos ofrecer a los usuarios de nuestras aplicaciones, es que puedan seleccionar de una manera fácil la impresora a utilizar, además de darles la posibilidad de que ajusten sus preferencias de impresión. Todo esto lo podríamos realizar manualmente creando un formulario que utilice un objeto del tipo *PrinterSettings* y que le proporcione a los usuarios las opciones que creamos convenientes. Pero para que todo sea, digamos, más homogéneo y no tengamos que reinventar la rueda, podemos hacer que toda esta configuración y selección de la impresora a utilizar la ofrezcamos mediante los mismos cuadros de diálogo que utiliza el resto de aplicaciones, y que se basan en los cuadros de diálogos comunes del sistema operativo. Si ésta es nuestra elección, nos resultará fácil, ya que podemos emplear la clase *PrintDialog*. Con un objeto de esta clase, tal como hemos comentado, es posible mostrar el mismo cuadro de diálogo que utilizan el resto de aplicaciones de Windows. Y con la ventaja añadida de que podemos configurar las opciones que se mostrarán, de forma que se adapte, en la medida de lo posible, a las preferencias de nuestra aplicación.

En breve veremos cuáles son las opciones que podemos usar para ajustar este cuadro de diálogo a nuestras preferencias o a las de nuestra aplicación.

LAS CLASES PARA IMPRIMIR EN .NET

Como hemos comentado en párrafos anteriores, existen varias clases que nos permitirán tanto imprimir, como configurar la impresión. Todo esto es posible gracias a las clases que .NET Framework pone a nuestra disposición para poder realizar esta tarea. Por desgracia, no disponemos del espacio suficiente para explicar con detalle todas las clases, delegados y enumeraciones que tanto el espacio de nombres *System.Drawing.Printing*, como *System.Windows.Forms* ponen a nuestra disposición para realizar tareas relacionadas con la impresión. Así que haremos una pequeña criba mostrando solamente los tipos que pueden resultarnos útiles.

La clase PrintDocument

Como hemos comentado anteriormente, ésta es la clase elemental o principal si queremos imprimir en .NET. De los miembros que ofrece, principalmente usaremos tres:

- El **método Print** es el que iniciará el proceso de impresión, haciendo que se produzcan los eventos necesarios para controlar lo que queremos imprimir.
- El **evento PrintPage**, que se producirá cada vez que tengamos que imprimir una página. Dentro de este evento es donde haremos todo lo necesario para imprimir cada una de las páginas, desde aquí podemos controlar si quedan más páginas por imprimir, etc.
- La **propiedad PrinterSettings**, a la que podemos asignarle un objeto del mismo nombre, en el que indicamos la impresora a usar y otros valores como el rango de página, el número de copias, el tamaño del papel, la orientación, etc.

Del resto de miembros de esta clase podemos destacar la propiedad *DocumentName*, a la que podemos asignar el nombre del documento que estamos imprimiendo y que será el que se muestre en el estado de la impresión. También tenemos el evento *QueryPageSettings*, que se produce justo antes del evento *PrintPage* y que nos permite asignar valores "particulares" a cada una de las páginas a imprimir antes de que se produzca este último evento.

La clase PrinterSettings

Tal y como hemos estado indicando en párrafos anteriores, esta clase nos permite especificar características de impresión como la impresora a usar, el número de copias a imprimir, el rango de páginas, etc. Realmente no necesitamos especificar nada para utilizar un objeto de esta clase, al menos de forma explícita, ya que al crear una nueva instancia los valores predeterminados serán los que tengamos asignado en la impresora, valga la redundancia, predeterminada de nuestro sistema. Pero si queremos modificar esos valores, tendremos que asignarlos a las propiedades de esta clase, entre las que podemos destacar las siguientes:

- **Copies** indica el número de copias a imprimir.
- **FromPage** indica la página a partir de la que queremos imprimir.
- **MaximumPage** indica el número máximo de copias.
- **MinimumPage** es el número mínimo de copias.
- **ToPage** indica la página hasta la que queremos imprimir

Tanto *MaximumPage* como *MinimumPage* se utilizarán para señalar los valores máximo y mínimo de las páginas disponibles, y se usarán con una clase del tipo *PrintDialog*.

Del resto de propiedades también podemos destacar las colecciones:

- **InstalledPrinters** es una propiedad compartida y, por tanto, podemos usarla sin necesidad de crear una instancia de la clase *PrinterSettings*. Esta colección devuelve un array de tipo *String* con el nombre de cada una de las impresoras instaladas en el sistema.
- **PaperSizes** es una colección con elementos del tipo *PaperSize* que nos permite saber los tamaños de papel que la impresora soporta. Cada elemento del tipo *PaperSize* nos proporciona información sobre el tamaño (ancho y alto), el nombre y la clase de papel, que no es ni más que una enumeración del tipo *PaperKind*.
- **PrinterResolutions** es una colección con elementos del tipo *PrinterResolution*, de forma que podamos averiguar las resoluciones (y calidades) permitidas por la impresora. Cada uno de estos elementos nos indicará tanto la resolución horizontal como la vertical, además de la calidad de impresión, especificada con uno de los valores de la enumeración *PrinterResolutionKind*: *Custom*, *Draft*, *High*, *Low* y *Medium*.

De los métodos, posiblemente el que más nos puede interesar es *CreateMeasurementGraphics*, que nos permite conseguir un objeto del tipo *Graphics* con el cual podemos averiguar ciertas características de la impresora sin necesidad de tener que imprimir. Esto se debe a que el objeto devuelto por este método es igual al que se incluye en la

clase *PrintPageEventArgs*, usada como segundo parámetro en los eventos producidos por la clase *PrintDocument*.

Como regla general, deberíamos tener una variable del tipo *PrinterSettings* para usarla como almacenamiento de las preferencias de impresión de nuestros usuarios, ya que esta clase se utiliza tanto con *PrintDocument* como con *PrintDialog*.

La clase PrintDialog

Esta clase nos servirá para que los usuarios seleccionen la impresora a emplear, así como para que indiquen ciertas características relacionadas con la impresión, como la calidad del papel, el número de copias, etc. La ventaja es que todo esto lo haremos utilizando el mismo cuadro de diálogo común incluido en Windows y empleado por la práctica totalidad de aplicaciones del sistema, tal como podemos comprobar en la Figura 6.

En ella podemos observar que hay ciertos elementos que puede que no nos interese mostrar o dejar habilitados, por ejemplo, si nuestra aplicación no permite imprimir el código seleccionado, no tiene mucho sentido ofrecer la opción *Selección*, (la figura está capturada de un Windows XP en inglés). Lo mismo ocurre con el botón de ayuda o con las cajas de texto que permiten indicar el rango de páginas a imprimir. Podemos habilitar o deshabilitar todas estas características mediante algunas de las propiedades de la clase *PrintDialog*, tales como *AllowPrintToFile*, *AllowSelection*, *AllowSomePages*, *PrintToFile*, *ShowHelp* y *ShowNetwork*

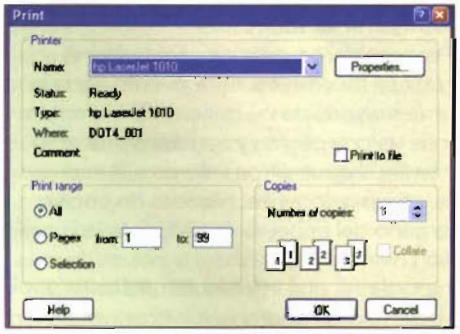


Figura 6. El cuadro de diálogo para seleccionar la impresora.

Al igual que en el resto de cuadros de diálogos de .NET, tenemos el método *ShowDialog* que será el que nos permita mostrar el cuadro de diálogo y saber si el usuario ha pulsado en el botón "OK" (Aceptar) o "Cancelar", para que de esta forma sepamos si debemos seguir con el proceso de impresión o no.

La propiedad *PrinterSettings* será la que nos permita conocer lo que el usuario ha seleccionado, además de ser la que utilicemos para asignar los valores predeterminados o bien los que tengamos almacenados de ocasiones anteriores. Tal como indicamos anteriormente, lo habitual será que tengamos una variable del tipo de esta propiedad con las pre-

Listado 6

```
Private Function seleccionarImpresora() As Boolean
    ' Devuelve True si todo fue bien o false si se canceló
    Dim prtDialog As New PrintDialog
    If prtSettings Is Nothing Then
        prtSettings = New PrinterSettings
    End If
    '
    With prtDialog
        .AllowPrintToFile = True
        .AllowSelection = True
        .AllowSomePages = True
        .PrintToFile = False
        .ShowHelp = True
        .ShowNetwork = True
        '
        prtSettings.MinimumPage = 1
        prtSettings.MaximumPage = 99
        prtSettings.FromPage = 1
        prtSettings.ToPage = 99
        prtSettings.Collate = False
        '
        .PrinterSettings = prtSettings
        If .ShowDialog() = DialogResult.OK Then
            prtSettings = .PrinterSettings
        Else
            Return False
        End If
    End With
    '
    Return True
End Function
```

Cuadro 2. La ayuda del cuadro de diálogo Imprimir

Como curiosidad, podemos decir que la propiedad *ShowHelp* nos permite indicar si se debe mostrar o no el botón de ayuda, (no la interrogación de la barra de títulos que siempre estará funcional). Pero entre las propiedades de esta clase no tenemos ninguna a la que indicarle qué ayuda se debe mostrar si pulsamos en ese botón. En lugar de ello, de lo que disponemos es de un evento, *HelpRequest*, que se producirá cuando el usuario pulse en dicho botón. Por tanto, si queremos mostrar algún tipo de ayuda, tendremos que usar ese evento para visualizar la información que creamos conveniente para nuestro cuadro de diálogo.

Nota

Por regla general, deberíamos asignar a la propiedad *Document* de la clase *PrintPreviewDialog*, el mismo objeto *PrintDocument* empleado para imprimir ya que la clase *PrintPreviewDialog* se encargará de que se produzcan los mismos eventos que si hubiésemos llamado al método *Print* del objeto *PrintDocument* asignado. De forma que lo que se muestre mediante este diálogo sea lo mismo que se imprima, que es al fin y al cabo lo que queremos conseguir.

ferencias del usuario. Por tanto, antes de llamar al método *ShowDialog*, deberíamos asignar a esta propiedad la variable que tengamos con las preferencias del usuario, y si no se ha cancelado, asignar nuevamente el resultado de dichas preferencias, tal como mostramos en el Listado 6 de la página anterior.

La clase PrintPreviewDialog

Esta clase nos permitirá mostrar una ventana con la vista preliminar del documento que queremos imprimir, de forma que los usuarios de nuestra aplicación puedan ver lo que se imprimirá. Dado que esta clase al estar derivada de *Form* tiene todas las propiedades, métodos y eventos de cualquier formulario, además de los relacionados con la previsualización del documento a imprimir, vamos a ver solamente los dos miembros que más nos interesarán.

- El **método ShowDialog** será el que se encargue de mostrar el formulario con la vista preliminar.
- A la **propiedad Document** le asignaremos un objeto del tipo *PrintDocument* que será el que utilicemos para saber qué es lo que queremos imprimir.

Tanto el método *Print* de la clase *PrintDocument*, como la clase *PrintPreviewDialog* utilizan los mismos eventos del objeto *PrintDocument*, por tanto, podríamos usar un método genérico que sea el encargado de mostrar una vista preliminar de lo que queremos imprimir o de mandarlo a la impresora. De este modo es posible emplear de forma común las opciones ofrecidas al usuario, como por ejemplo, permitir la selección de la impresora antes de imprimir, etc.

En el Listado 7 vemos cómo podría ser ese método genérico para elegir entre imprimir o previsualizar lo que deseamos imprimir.

Como podemos ver en la Figura 7, el formulario (o cuadro de diálogo) de previsualización nos permite seleccionar el número de páginas a mostrar, si queremos ver una o más al mismo tiempo, el porcentaje de ampliación e incluso imprimir lo que estamos viendo, todos estas características ya están incluidas en ese formulario.

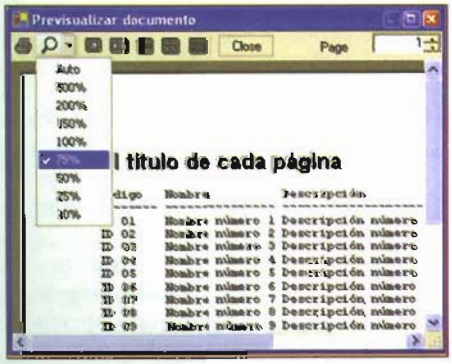


Figura 7. El formulario de previsualización.

La clase PrintPreviewControl

Si nuestra intención es crear nuestro propio formulario de previsualización, también podemos hacerlo utilizando el control *PrintPreviewControl*, que es el que usa la clase *PrintPreviewDialog*, si bien tendremos que crear todos los botones y opciones. Para ello usaremos los miembros específicos de este control, tales como:

- **AutoZoom** lo usaremos para que al cambiar el tamaño del control se modifique también la página mostrada.
- **Columns** indica el número de páginas a mostrar cuando se elija la orientación horizontal (apaisada).
- **Document** es donde asignaremos el objeto *PrintDocument* a imprimir.
- **Rows** indica el número de páginas a mostrar cuando elijamos la orientación vertical.
- **Zoom** para indicar la ampliación con la que queremos mostrar los documentos.
- **StartPageChanged** en un evento que se producirá cada vez que cambiemos la página de inicio (este evento nos servirá para crear un equivalente al *NumericDropDown* usado en la clase *PrintPreviewDialog*).

Si también quisiéramos implementar un botón para imprimir, tendremos que manejar nosotros mismos la impresión. Sin embargo, resultará fácil, ya que lo único que tendríamos que hacer es llamar al método *Print* del objeto *PrintDocument* asignado a la propiedad *Document*.

Como siempre, la última palabra la tendremos nosotros y, dependiendo de lo que queramos hacer, usaremos una clase u otra.

PARA TERMINAR, UN POCO MÁS ALLÁ

Por supuesto, ni la presentación ni la impresión de los datos que necesitamos mostrar son procesos automáticos o sencillos, al menos si queremos darles un toque profesional. Seremos nosotros los

Listado 7

```
Private Sub imprimir(ByVal esPreview As Boolean)
    ' imprimir o mostrar el PrintPreview
    '
    If prtSettings Is Nothing Then
        prtSettings = New PrinterSettings
    End If
    '
    If chkSelAntes.Checked Then
        If seleccionarImpresora() = False Then Return
    End If
    '
    If prtDoc Is Nothing Then
        prtDoc = New PrintDocument
        AddHandler prtDoc.PrintPage, AddressOf prt_PrintPage
    End If
    '
    ' la línea actual
    lineaActual = 0
    '
    ' la configuración a usar en la impresión
    prtDoc.PrinterSettings = prtSettings
    '
    If esPreview Then
        Dim prtPrev As New PrintPreviewDialog
        prtPrev.Document = prtDoc
        prtPrev.Text = "Previsualizar documento"
        prtPrev.ShowDialog()
    Else
        prtDoc.Print()
    End If
End Sub
```

que debamos "afinar" la forma de mostrar esos resultados. La ventaja que nos ofrece .NET es que tenemos posibilidades de hacerlo de una forma más o menos fácil y que, de alguna manera, nos facilita bastante la tarea.

Aunque para la mayoría de situaciones será más que suficiente, podemos aprovechar las posibilidades que nos proporciona la POO para crear nuestras propias clases e implementar metodos de impresión que se adapten a los datos a mostrar. Una de las claves de la POO es la abstracción, y qué mejor forma de emplear la abstracción que crear un método que se encargue de manejar los datos que la propia clase mantiene, y que "sepa" cómo mostrar esos datos e incluso qué formato debe aplicarse a la hora de presentarlos por impresora.

Seguro que el lector puede pensar que no es necesario ese esfuerzo extra para imprimir una línea, y probablemente tenga razon, pero precisamente si tenemos en cuenta estos detalles cuando diseñamos nuestras clases para manejar datos, nos resultará más fácil realizar cambios en el futuro.

Por ejemplo, si tenemos una clase en la que existen datos de cadena y numéricos y queremos mostrarlos, la forma de tratar cada uno de esos datos será diferente, ya que querremos ajustar los datos numericos a la derecha y los de cadena a la izquierda, además de que si debemos recortar la información, con total seguridad preferiremos "truncar" los datos de cadena antes que los numéricos. En este tipo de situaciones si dejamos que el código de la clase sea el que decida estos truncamientos, siempre será preferible que tener que hacerlo en el propio evento de impresión. Si las clases utilizadas para contener los datos están preparadas para imprimir el contenido de cada línea, resultará tan sencillo como llamar a un método de cada una de esas clases.

En el **Listado 8** podemos ver cómo quedaría el evento *PrintPage* si hiciéramos algo de lo que acabamos de comentar.

El código completo se incluye en el CD-ROM de la revista, y hemos definido un interfaz con cuatro métodos: una colección para almacenar objetos del tipo del interfaz; una clase abstracta que define los dos métodos usados en el evento mostrado en el **Listado 8**; tres clases diferentes para ver cómo poder usar tanto la clase abstracta como el interfaz, dos de ellas se basan en la clase abstracta (por tanto parte del código ya estará disponible) y la tercera simplemente implementa el interfaz y define los cuatro métodos; y por último, también incluimos un formulario que podemos utilizar de forma genérica con cualquiera de esas tres clases o de cualquier otra que utilice el interfaz o la clase abstracta. De esta forma, el lector podrá constatar que no todo son palabras, y además, para comprobar la utilidad de ese esfuerzo extra, también incluimos otros tres proyectos, dos de los cuales imprimen el contenido de un *ListView*, uno de ellos utilizando un código parecido al proyecto que emplea el interfaz

CONCLUSIÓN

Tal como hemos podido comprobar, imprimir en .NET no es tan complicado como en un principio pudiera parecer. Además, tenemos valores añadidos que nos permiten un mayor control sobre la impresión y, especialmente, sobre las opciones que podemos ofrecer a los usuarios de nuestras aplicaciones. Como hemos visto, hacer algo como la presentación preliminar, que en otros lenguajes nos obligaría a escribir bastante código, es tan sencillo como crear un nuevo objeto en la memoria y asignar un par de propiedades. (Ejemplos incluidos en el CD-ROM).

Listado 8

```
Private Sub prt_PrintPage(ByVal sender As Object, ByVal e As
PrintPageEventArgs)
    Dim lineHeight As Single
    Dim yPos As Single = e.MarginBounds.Top
    Dim leftMargin As Single = e.MarginBounds.Left
    Dim printFont As System.Drawing.Font
    '
    ' Asignar el tipo de letra
    printFont = prtFont
    lineHeight = printFont.GetHeight(e.Graphics)
    '
    ' imprimir la cabecera de la página
    yPos = Datos(0).CabeceraImpresión(e, printFont, yPos)
    '
    ' imprimir cada una de las líneas de esta página
    Do
        yPos += lineHeight
        e.Graphics.DrawString( _
            Datos(lineaActual).LineaImpresión, _
            printFont, Brushes.Black, leftMargin, yPos)
        lineaActual += 1
    Loop Until yPos >= e.MarginBounds.Bottom _
        OrElse lineaActual >= Datos.Count
    '
    If lineaActual < Datos.Count Then
        e.HasMorePages = True
    Else
        e.HasMorePages = False
    End If
End Sub
```

Resumen

Aunque cada plataforma utiliza su propia visión de cómo debería imprimirse correctamente, ambos métodos consiguen finalmente el resultado que deseamos. Ninguno de ellos nos libra del arduo trabajo de preparación gráfica de los datos, pero al menos nos escudan de las interioridades del hardware de impresión que se encuentra conectado al equipo o la red local. En ambos casos, un par de pruebas bastarán para que nos familiaricemos con las metodologías que hemos expuesto y tener acceso a impresiones básica para comenzar. Con algo de trabajo adicional podremos generar informes visualmente muy atractivos. Así que: ¡Feliz impresión!

1ª entrega, CD-ROM y Presentación de la obra por solo 5,99 euros

Programación y Diseño WEB

La primera obra para aprender a programar y diseñar sitios web completos

- Cómo programar sitios web con los lenguajes Java, JavaScript y PHP. Además cómo realizar consultas con la base de datos MySQL
- Aprende a trabajar con los mejores programas de diseño web: Dreamweaver, Flash y Fireworks

Composición de la obra

12 coleccionables

divididos en:

Programación web
> Curso de programación práctico de JavaScript y Java
> Domina el gestor de bases de datos MySQL
> Desarrollo web con PHP

Diseño web
Sección dedicada íntegramente a la suite Macromedia Studio MX 2004
> Dreamweaver MX 2004
> Flash MX 2004
> Fireworks 2004

Tutorial práctico
Sección práctica donde a lo largo de las doce entregas se realiza una web completa de ejemplo que incluye comercio electrónico y acceso a bases de datos. Todo siguiendo los conceptos explicados en las secciones de Programación y Diseño.

Contenido CD-ROM
Descripción del contenido del CD-ROM y explicación de cómo instalar el software.

12 CD-ROMs

En cada CD-ROM que acompaña al coleccionable se encuentra el material necesario para seguir el curso: código fuente de los ejemplos, imágenes para las secciones de diseño, archivos para el tutorial, etc. Además programas completos y demos comerciales de las mejores aplicaciones actuales relacionadas con la creación de páginas web.

Todos los CD-ROMs incorporan un asistente para mayor comodidad en la búsqueda de contenidos e instalación de programas.

C/ del Río Ter, Nave 13. Polígono "El Nogal"
28110 Algete (Madrid)
Tel.: 916280203. Fax: 916280935
e-mail: video@iberprensa.com
www.iberprensa.com

ya a la venta **semanalmente** en quioscos y centros comerciales