Año 1 • Número 10 • **6,50** euros

Todo Programación

La Revista bimestral para entusiastas de la programación

www.iberprensa.com

Instalación de Mono paso a paso

Gráficas en Java con JFreeChart

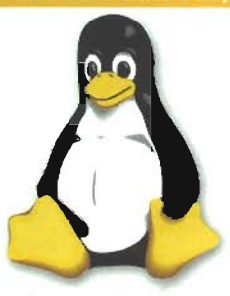
Nuevo curso de programación Linux con GTK#

Creación de librerías DLLs desde Delphi

CD-ROM: MONO 1.0.5 PARA LINUX Y WINDOWS, OPENTORO 1.0 Y HERRAMIENTAS PARA PROGRAMADORES

Zona Linux

- Mono Código inseguro en C#
- Patentes y programación de aplicaciones



Zona Windows

- Examinando ensamblados con Reflection
- Utilizando la herramienta FxCop

TECNOLOGÍA GRID: ¿QUÉ ES? ¿QUIÉN HAY DETRÁS? VENTAJAS E INCONVENIENTES

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)

DISEÑO PORTADA
Antonio Gª Tomé

MAQUETACIÓN
Antonio Gª Tomé

DIRECTOR DE PRODUCCIÓN
Carlos Peropadre
cperopadre@iberprensa.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
(Añada 34 si llama desde fuera de España.)

DEPÓSITO LEGAL: M-13679-2004
Número 10 • Año 1

EDITORIAL

Eduardo Toribio

Nuevos tiempos

Queriendo seguir siendo fieles a nuestra vocación de ofrecer temas que combinen el contenido didáctico con la actualidad del sector, incluimos en este número un doble cover dedicado a la tecnología Grid por un lado y al entorno de desarrollo Mono por otro. Ambos son dos de los asuntos que están más en boca de todos dentro de la comunidad de desarrolladores Windows/Linux. Por tanto, creo que este número os resultará muy interesante, además incorporamos una minisección dedicada a la librería GTK# y varios tutoriales completos relacionados con .NET y la programación para Windows.

Lamentablemente, las circunstancias del mercado han ocasionado que tengamos que incrementar en 50 céntimos el precio de portada, ya que los lectores son el único medio de financiación de una revista técnica independiente como es esta que tenéis en vuestras manos. Esperamos vuestra comprensión ante tal medida, y trataremos de, número a número, entregaros una revista lo más provechosa posible que seguramente justificará el precio actual.

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

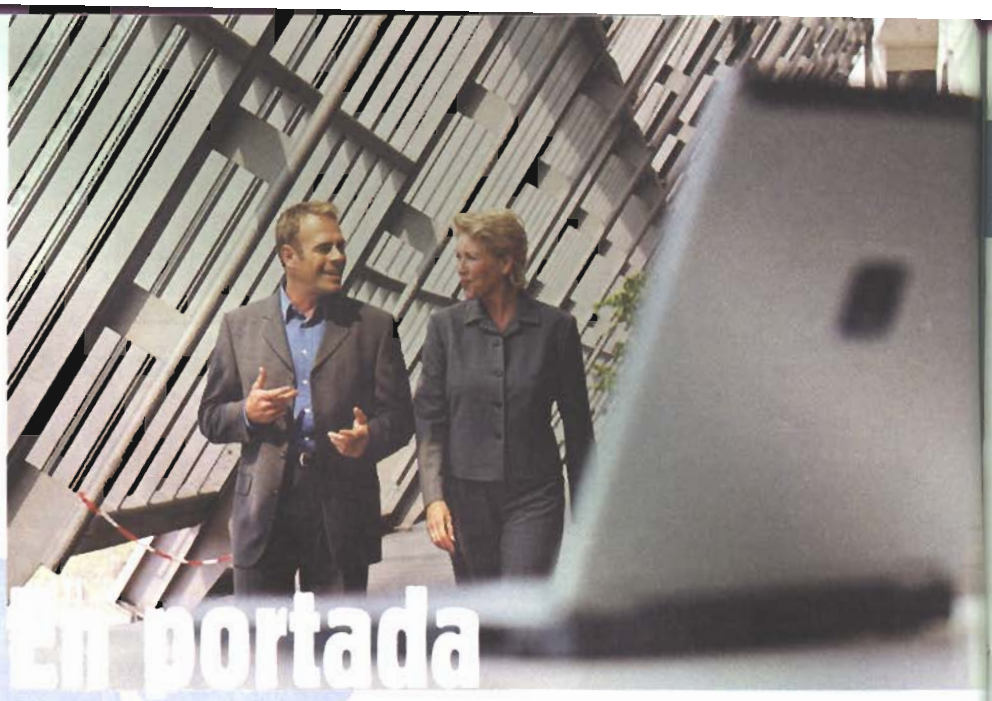
Si le interesa conocer nuestras tarifas de publicidad no dude en ponerse en contacto con nuestro departamento comercial:
- **Tel. 91 628 02 03**
- **e-mail: publicidad@iberprensa.com**

Sumario

Número 10

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.



En portada

Introducción a la tecnología Grid

11 La tecnología Grid está comenzando a hacer mucho ruido: grandes empresas como IBM afirman que desarrollan grids, y otras como Oracle que sus productos están orientados a esta tecnología. Veremos en este amplio reportaje en qué consiste, sus ventajas e inconvenientes, y qué es lo que realmente hay de Grid en las propuestas actuales de las empresas.

ZONA LINUX

Instalación del proyecto Mono >>

16 Por temas de patentes no es habitual que las distribuciones incluyan Mono, por tanto es necesario realizar la instalación manualmente. Vamos a ver este proceso detenidamente y paso a paso.

Código inseguro en C# >>

20 Estudiamos en esta ocasión la manera de implementar código inseguro en C#, es decir, los clásicos punteros de C++. Veremos las particularidades de la plataforma .NET y algunos ejemplos prácticos.

Curso de programación con GTK# >>

24 Comenzamos una serie de artículos para aprender a programar utilidades gráficas para escritorio mediante la librería GTK#, el objetivo del curso es poder desarrollar nuestros propios programas para distintas plataformas.

Ejemplos y código fuente

Cada CD-ROM de la revista incluye una carpeta denominada *fuentes* en la que se encuentra el material complementario para seguir cada uno de los cursos: los listados completos, los ejemplos desarrollados en diversos lenguajes, compiladores, editores, utilidades y en general, cualquier herramienta que se mencione o cite en la respectiva sección.
Todo con la finalidad de completar la formación y facilitar el seguimiento de cada artículo por parte del lector.

CONTENIDO DEL CD-ROM

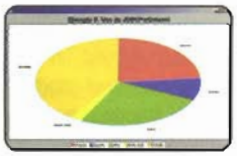
Herramientas y recursos para el programador

64 En el CD de **Todo Programación** incluimos este mes la última versión de Mono al completo, para que puedas desarrollar para .NET desde cualquier plataforma, ya sea Windows, UNIX o Linux, y el Kit Java Application Verification, que te permitirá elevar el nivel de calidad de tus programas en Java. Completan la oferta herramientas como FxCop para el análisis de código, el gestor de contenidos web OpenToro o el entorno de desarrollo PL/SQL Developer.

TALLER PRÁCTICO

Creación de DLLs desde Delphi >>

37 Vamos a aprender a utilizar las librerías DLL ya existentes en Windows, para pasar después a crear las nuestras propias. Utilizaremos para ello el entorno de desarrollo Borland Delphi.



Construcciones de gráficas en Java con JFreeChart >>

42 Dentro de las distintas librerías Java que encontramos para la creación de gráficos destaca JFreeChart, que es además software libre. Veremos en este tutorial cómo se instala y se usa.

Criptografía y seguridad informática >>

47 Terminamos nuestra miniserie de criptografía examinando las vulnerabilidades de la criptografía y sus posibles soluciones a través de PKI. Estudiaremos los certificados y las autoridades de certificación.

Y ADEMÁS...

Juegos Java: El juego del pistolero >>

55 Utilizaremos el desarrollo del juego del pistolero como ejemplo para asentar todos los conocimientos vistos hasta este momento en la serie. Además se introducirán nuevos conceptos en este ejemplo mucho más completo que los anteriores.

.net ZONA WINDOWS

Utilizando la herramienta FxCop >>

29 FxCop es una herramienta para .NET Framework, en concreto se puede utilizar para analizar los ensamblados con el fin de comprobar si siguen las recomendaciones de diseño de la biblioteca de clases de .NET.

Examinando ensamblados con Reflection >>

33 Reflection es otra herramienta poderosa para el programador .NET ya que permite utilizar los metadatos de los ensamblados dentro del propio código, sin necesidad de archivos externos de cabecera como en C++.

NOTICIAS

6. **Nuevo** sistema europeo de telecomunicaciones móviles Sailor.
6. **IBM** cede 500 patentes para software libre.
7. **Acuerdo** tecnológico entre Sun y la URJC.
7. **Dell**, EMC, Intel y Oracle impulsan Grid Computing.
8. **Informe** METAspectrum.
8. **Sun** lanza Ray Server Software 3.0 y Ray 170.
8. **SignBank**.
8. **Nuevo** Intel Centrino.
9. **Archos** PMA400.
9. **Portátil** Toshiba Tecra A2.
9. **Pantalla** TFT Yamada L171.

Bases de datos: Triggers en PL/SQL >>

60 Entramos en la recta final de nuestro curso de PL/SQL estudiando los conceptos que aún no hemos analizado de este importante lenguaje procedural. Entre ellos destacaremos los triggers, elementos fundamentales en las bases de datos modernas.

NIVEL DE DIFICULTAD:

MATERIAL INCLUIDO EN EL CD-ROM:
Ejemplos ■ FreeChart 0.9.14

Creación de gráficas en Java con JFreeChart

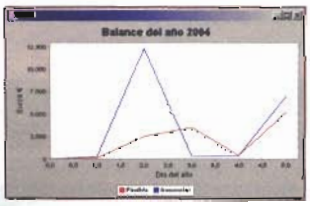
MANUEL DOMÍNGUEZ
mdominguez@iberprensa.com

La informática es ya una realidad palpable en todas las empresas del mundo. Antes los empresarios requerían una buena automatización de sus cálculos para ahorrar trabajo, ahora eso se da por supuesto, y lo que piden es que todo sea intuitivo, esté a un par de clics de ratón y sobre todo, gráficas estadísticas de absolutamente todo. Por tanto, ante esta demanda, hay que estar preparados para convertirse en oferta.

Del texto se pasó al gráfico, y del teclado y sus combinaciones de teclas, al ratón o al touchpad de los portátiles. El siguiente paso que debía da una aplicación era, lógicamente, aprovechar estas características para ofrecer una visión del negocio sencilla y rápida. Esta situación es extrapolable a todos los ámbitos: programas de gestión de redes, antivirus, software de monitorización de sistemas, Kazaa, Shareazaa, etcétera, muestran de forma gráfica y de un plumazo toda la información que el usuario quiere ver; hoy en día se buscan pantallas que resuman todo.

Exigencias de los clientes

¿Qué pide un cliente? Pues generalmente todo lo que le podamos proporcionar por el precio que está dispuesto a pagar, lo vaya a usar o no. Pero, sin embargo, cuando se habitúa al software que se le ha programado, la mayoría coinciden en comenzar a pedir una serie de cosas concretas: ayudas de teclado, búsquedas directas en los formularios de entrada, autocompletado de datos, búsquedas por campos e, irremediablemente, gráficas donde lo pueda ver todo sin abrumarse por ingentes cantidades de números; gráficas de tendencia, ganancias, pérdidas, estimaciones futuras ... y que además, por supuesto, todo eso lo pueda imprimir, guardar, etcétera.



Ejemplo de gráfica pedida por un cliente.

Amigabilidad

Y no basta con que se le programen las gráficas. El cliente también exige (ya que lo paga) que sea fácil de mostrar, que no le estorbe tapándole el lugar de trabajo habitual, que la pueda redimensionar y minimizar, que tenga barras de desplazamiento, que cuando contenga muchos datos no aparezca todo comprimido, que puedan realizar zoom de la zona que les interese y comparar datos entre gráficas de años distintos, en definitiva, ¡todo!

Diversidad de opciones

Afortunadamente hay innumerables productos en el mercado que permiten crear gráficas de forma rápida y añadirlas a nuestra aplicación. Dentro del software libre, también tenemos diversas librerías para Java que permiten hacerlo: Chart2D, JRobin, JFreeChart, etc. Nos centraremos en esta última. Ya sabemos qué hace, para qué sirve y con qué lenguaje de programación la podemos usar: Java. De aquí en adelante veremos cómo descargarla, instalarla y usarla para implementar en muy poco tiempo un sistema de gráficas para nuestras aplicaciones.

LA LIBRERÍA JFREECHART

JFreeChart es una librería gráfica con licencia LGPL que forma parte de un proyecto liderado por David Gilbert. Para poder usar esta librería, debemos obtener de la web dos paquetes imprescindibles: *jfreechart-xxx.zip* y *jcommon-xxx.zip* (se descargan juntos en un único archivo) donde *'xxx'* habrá que sustituirlo en cada caso por la versión más avanzada. La web para descargar estos dos paquetes es *http://www.jfree.org*, donde además hay otros productos disponibles. La página de descarga nos llevará a SourceForge que es donde está alojado el proyecto. En el CD-ROM incluimos la librería versión 0.9.14.

Instalación

Una vez que tenemos los dos ficheros descritos en el apartado anterior, lo único que debemos hacer es descomprimirlos. Para cada uno se creará una estructura de directorios y en la raíz de ambas tendremos los dos ficheros *.JAR que nos interesan: *jfreechart-xxx.jar* y *jcommon-xxx.jar*. Lo que debemos hacer para poder empezar a usar la API de JFreeChart es incluir esos dos ficheros JAR en la variable de entorno *CLASSPATH* y fin de la instalación. Sencillo ¿verdad? pues ya solo queda conocer la API y a partir de ahí podremos llenar de gráficas nuestra aplicaciones. A lo largo de todo el artículo vamos a suponer que trabajamos con Netbeans IDE, puesto que es un tema que se ha tratado muchas veces en **Todo Programación** y este software se ha incluido en el CD en diversas ocasiones. No obstante, se puede trabajar con cualquier IDE, ya que lo que haremos funcionará en cualquier entorno que tenga JSDK 1.4.x o superior. Supondremos que sí está instalado y funcionando correctamente.

Series

Una serie de datos es el conjunto de datos de uno de los valores de la gráfica. Si estamos representando dos cosas en una misma gráfica, por ejemplo, ganancias y pérdidas, las ganancias constituirán una serie y la pérdidas otra. Generalmente cada serie se representa en un color distinto. En las elecciones del gobierno, en las gráficas que representan el número de votos por partido, cada partido es una serie. En JFreeChart tenemos varias clases para representar las series, como podemos observar en la figura de la página siguiente, que atienden a las particularidades de cada tipo de gráfico.

De entre todas las implementaciones existentes, las más usadas son *XYSeries* y *TimePeriodValues*. La primera sirve para almacenar datos para la gráfica en forma de puntos (x,y) que serán mostrados. La segunda nos permite generar rangos temporales con significado en "el eje de las

equis", por ejemplo, días de la semana, meses del año, hora del día, entre otras. Son los dos tipos de series más usados. Para crear una nueva serie, el constructor más sencillo es el que especifica el nombre de la serie, por ejemplo:

```
XYSerie miSeriel = new
XYSerie("Todo Programación");
TimePeriodValues miSerie2 = new
TimePeriodValues("Todo Programación");
```

Y éste será el nombre que aparecerá en la leyenda de la gráfica, una vez que esté generada para esta serie. En este caso, las series mostrarían los valores para la revista **Todo Programación**.

En cualquier caso, necesitamos poder añadir y actualizar valores en las series. Para ello todas las series incorporan un método *add()* y otro *update()*, aunque según el tipo de serie, los parámetros de estos métodos varían. Para las series de tipo *XYSerie*, el método más usado es *add(Number x, Number y)*, o sea, los dos parámetros son un número, por ejemplo:

```
miSeriel.add(23, 45)
```

Establecería un nuevo valor (x,y); concretamente el punto (23,45). Y para actualizar este valor, se usa el método *update (Number x, Number nuevaY)*. De nuevo dos números. En este caso, la siguiente línea:

```
MiSeriel.update(23, 56)
```

Modificaría el valor introducido unas líneas más arriba, es decir, cambiaría el par (23, 45) por el par (23, 56).

Para series de tipo *TimePeriodValues*, el método *add()* más común es *add (TimePeriod p, Number valor)*, es decir, el primero de los parámetros especifica el periodo de tiempo, y el segundo un número, el valor.

Así la siguiente línea:

```
miSerie2.add(UnPeriodo, 34)
```

Añadiría un nuevo valor a la serie, concretamente añadiría el valor 34 en el periodo *UnPeriodo*. La forma de actualizar este valor es un poco más extraña. Se usa el método *update(int indice, Number nuevoValor)*, que lo que hace es actualizar el *TimePeriod* de la serie cuyo índice de entrada en la serie es *indice*, y ponerle el valor *nuevoValor*. Por ejemplo:

```
miSerie2.update(1, 55)
```

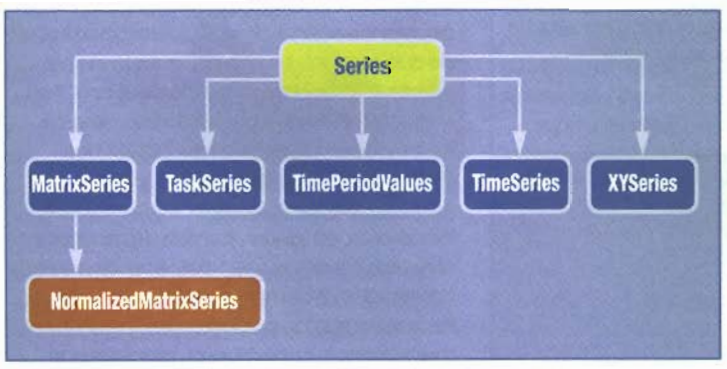
Cambiaría el valor 34 por 55 en la serie *miSerie2*, siempre que el índice de entrada de *unPeriodo* en la serie sea 1. Así, en cualquiera de los casos, iremos conformando el conjunto de datos de la serie que debe representar la gráfica, por ejemplo, los puntos de la función matemática Seno, para x desde 0 hasta 150, o el número de ventas de melones por cada día de la semana.

Lo anterior está bien pero... ¿Qué es un *TimePeriod*? Es un objeto que representa un espacio determinado de tiempo: una hora, un día, un año, un segundo... En la documentación Javadoc que acompaña a JFreeChart, se puede aprender más. Simplemente comentaremos algunas de las implementaciones más usuales de *TimePeriod*.

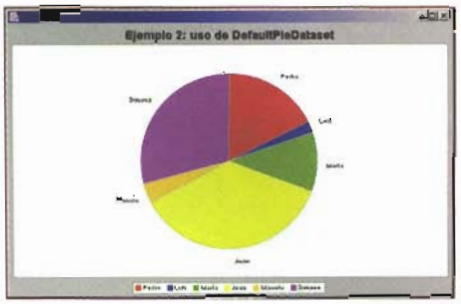
- **Hour.Hour(int hora, int dia, int mes, int año)**, permite crear un *TimePeriod* que representa una hora concreta.
- **Day.Day(int dia, int mes, int año)**, permite crear un *TimePeriod* que representa un día concreto.
- **Month(int mes, int año)**, permite crear un *TimePeriod* que representa un mes concreto del año.
- **Year.Year(int year)**, permite crear un *TimePeriod* que representa un año concreto.

Conjunto de series

Ya hemos creado una serie, aunque lo común es representar en una misma gráfica varias series de datos. Las gráficas de JFreeChart no trabajan directamente con series, sino con conjuntos de ellas. Si vamos a representar varias series en una gráfica, debemos agruparlas en un conjunto y asignar ese conjunto a la gráfica, que se encargará de mostrar todo correctamente.



Tipos de series soportados en JFreeChart.

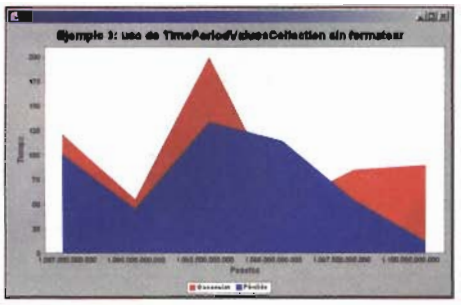


Ejemplo de gráfica de sectores básica.

Todos los conjuntos de series parten de la clase abstracta *AbstractDataset*. Aquí todo se complica bastante, puesto que el número de clases que parten de *AbstractDataset* parece no tener fin, y a sus subclases les ocurre lo mismo. Sin embargo, hay algunos conjuntos de series que se usan más que otros, como por ejemplo *DefaultPieDataset*, usada para gráficos de sectores; *XYSeriesDataset*, para series que representan pares de valores (x, y); *TimePeriodValuesCollection*, para series de tipo *TimePeriodValues*, etcétera. El número de posibilidades es bastante elevado, y no resulta demasiado sencillo en este caso saber qué conjunto de series se debe utilizar para cada ocasión. Ante la duda, lo mejor es echar un vistazo a los ejemplos que acompañan a JFreeChart, que son bastante intuitivos e incorporan el código fuente.

No todos los conjuntos de series implementan los mismos métodos. Por supuesto hay unos métodos comunes, derivados de la herencia, pero los importantes no son iguales en todos los conjuntos de series. Para los conjuntos de series, en casi cualquiera de sus implementaciones, el método más usado es *addSeries(unaSerie)*, donde *unaSerie* dependerá del tipo de conjunto de series utilizado. Si es un conjunto *TimePeriodValuesCollection*, será de tipo *TimePeriodValues*, si es un conjunto *XYSeriesCollection*, será de tipo *XYSeries*, etcétera.

Sin embargo, existen tipos de gráficos para los cuales el concepto de series no tiene sentido, por ejemplo, en los gráficos de sectores, puesto que sólo se pueden presentar valores en una dimensión. En estos casos los conjuntos de datos (por ejemplo *DefaultPieDataset*) suelen implementar un método llamado *setValue (Comparable key, Number value)* donde *key* suele ser el nombre de lo que queremos representar, por ejemplo "pimientos" y *value* es el valor asociado, por ejemplo



Ejemplo de áreas básica con periodos de tiempo.

54. En los ejemplos *ejemplo1.java*, *ejemplo2.java* y *ejemplo3.java* que se pueden encontrar en el CD que acompaña a la revista, se muestran ejemplos de uso de estas clases que estamos comentando.

Tipos de gráficos

Con todo lo que hemos visto hasta ahora, ya sabemos crear el conjunto de datos que se deben de representar en un gráfico. Concretamente hemos aprendido a realizarlo para gráficos de sectores, o para aquellos que presentan series de datos, sean continuos o por periodos de tiempo. Ahora, ese conjunto de datos se puede representar de muy diversas maneras: líneas, puntos, dispersión, áreas, barras... y un largo etcétera. Todos los gráficos de esta librería son instancias de la clase *JFreeChart*, que representa a cualquier gráfico de los que se pueden crear.

Existen varias formas de crear un gráfico a partir de un conjunto de datos. La primera es haciendo uso de los métodos *createXXXXXChart(...)* de la clase *ChartFactory*, que implementa un conjunto de métodos que crean de forma rápida y sencilla los gráficos más comúnmente utilizados. En el método, *XXXXX* representa el tipo de gráfico. Así, *createXYLineChar(...)* crearía un gráfico de líneas que une puntos (x,y); *createPieChart(..)* un gráfico de sectores; *createStackedBarChart(...)* crearía un gráfico de barras apiladas, etc.

De esta forma, es posible que con un mismo conjunto de datos podamos generar distintos gráficos. Sin embargo, no todos los conjuntos de datos permiten esto. Por ejemplo, no se puede representar como un gráfico de líneas un conjunto de datos para un gráfico de sectores, ni viceversa. Los parámetros del método *createXXXXXChart(...)* que usemos, serán quienes determinarán las posibilidades. En la Tabla 1 se muestran los métodos más utilizados de ChartFactory para crear los gráficos comunes.

Siguiendo esta forma de crear gráficos, habría que escribir algo como:

```
JFreeChart miGrafico =
ChartFactory.createXXXXXChart(…)
```

Y ya dispondríamos de un gráfico que mostraría, según el tipo que hayamos elegido, el conjunto de datos especificado, pero este gráfico no es directamente visible en un componente Swing de Java. Para poder mostrar el gráfico, tendremos que invocar al método *createBufferedImage(int alto, int ancho)* del objeto *JFreeChart*, o sea, de nuestro gráfico. Esto nos devolverá una imagen que, ahora sí, podremos mostrar en todos aquellos componentes Swing que lo permitan.

```
BufferedImage miImg =
miGrafico.createBufferedImage(300,
200);
```

La segunda forma de crear un gráfico para un conjunto de datos es mucho más complicada, pero permite tomar muchas decisiones acerca de qué queremos mostrar, de qué manera, etcétera. Consiste simplemente en crear una instancia de JFreeChart. El problema es que haciendo uso de esta forma debemos especificar TODOS los aspectos de un gráfico, como el título, el conjunto de datos, los ejes, las categorías, los colores, etiquetas, formas de mostrar en el gráfico los datos... en definitiva, demasiada información. Además, en la mayor parte de las ocasiones, los gráficos creados de la primera forma, y con algunos retoques que veremos, son más que suficiente.

Tabla 1. Distintos métodos createXXXXChart(...) de ChartFactory

Nombre del método	Parámetros del método (en orden)	Descripción
createAreaChart	String title: título del gráfico. String categoryAxisLabel: Nombre del eje X. String valueAxisLabel: Nombre del eje Y. CategoryDataset data: Datos en forma de categorías. PlotOrientation orientation: Orientación del gráfico. boolean legend: Mostrar leyenda o no. boolean tooltips: Mostrar comentarios emergentes o no. boolean urls: Permitir URL's o no (navegar al hacer clic).	Crea un gráfico de áreas para un conjunto de datos cuyas series son categorías, por ejemplo, naranjas, plátanos, sandías...
createBarChart	String title: título del gráfico. String categoryAxisLabel: Nombre del eje X. String valueAxisLabel: Nombre del eje Y. CategoryDataset data: Datos en forma de categorías. PlotOrientation orientation: Orientación del gráfico. boolean legend: Mostrar leyenda o no boolean tooltips: Mostrar comentarios emergentes o no. boolean urls: Permitir URL's o no (navegar al hacer clic).	Crea un gráfico de barras para un conjunto de datos cuyas series son categorías, por ejemplo, naranjas, plátanos, sandías...
createHistogram	String title: título del gráfico. String xAxisLabel: Nombre del eje X. String yAxisLabel: Nombre del eje Y. IntervalXYDataset data: Datos en forma de intervalos. PlotOrientation orientation: Orientación del gráfico. boolean legend: Mostrar leyenda o no. boolean tooltips: Mostrar comentarios emergentes o no. boolean urls: Permitir URL's o no (navegar al hacer clic).	Crea un histograma para un conjunto de datos cuyas series vienen dadas por intervalos, por ejemplo, 0-10 años, 11-25 años, >25 años...
createPieChart	String title: título del gráfico. CategoryDataset data: Datos en forma de categorías. Int extractType: Datos tomados por filas o columnas. boolean legend: Mostrar leyenda o no. boolean tooltips: Mostrar comentarios emergentes o no. boolean urls: Permitir URL's o no (navegar al hacer clic).	Crea un gráfico de sectores para un conjunto de datos cuyas series son categorías, por ejemplo, C++, Java, Delphi...
createXYLineChart	String title: título del gráfico. String xAxisLabel: Nombre del eje X. String yAxisLabel: Nombre del eje Y. XYDataset data: Datos en forma de pares (x,y). PlotOrientation orientation: Orientación del gráfico. boolean legend: Mostrar leyenda o no. boolean tooltips: Mostrar comentarios emergentes o no. boolean urls: Permitir URL's o no (navegar al hacer clic).	Crea un gráfico de líneas para un conjunto de datos cuyas series vienen dadas por pares (x,y), por ejemplo, (2,3), (23,1)...

Paneles específicos

El programa *ejemplo1.java* que se puede encontrar en el CD de la revista, crea un gráfico como se ha explicado hasta ahora. Sin embargo, esto tiene un problema, la imagen de un gráfico es estática y hay que refrescarla frecuentemente. Esto no es bueno para situaciones en las que se desee ver en tiempo real la evolución de un gráfico muy cambiante, por ejemplo, en un simulador. Además, la imagen tiene un tamaño fijo y no es factible ampliarla si no es generando otra copia nueva. Por ello, la librería JFreeChart incorpora una clase que extiende el jPanel de Swing. La forma de crear un *ChartPanel*, que es como se llama, es la siguiente:

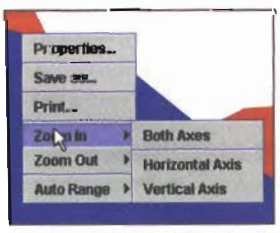
```
ChartPanel miPanel =
new ChartPanel(miGrafico);
```

Y con esta simple línea tendremos un panel gráfico que representa el gráfico JFreeChart que habíamos creado, pero que además, ofrece automáticamente muchas mejoras: redimensionamiento automático, regeneración automática del gráfico cuando se añaden datos a las series, menú de propiedades (clic derecho sobre el gráfico) que nos permite guardar como PNG, imprimir el gráfico, cambiar sus propiedades, hacer zoom, etcétera.

El creador de JFreeChart recomienda siempre el uso de ChartPanel para representar gráficos, y es un buen consejo, pues facilita todo enormemente, además de proporcionar opciones muy interesante y de forma automática. Tras obtener nuestro panel, hay que añadirlo al interfaz gráfico como haríamos con cualquier otro componente. Los ficheros *ejemplo2.java* y *ejemplo3.java* usan este método y el código fuente es realmente pequeño.

PERSONALIZACIÓN DE LOS GRÁFICOS

Ya sabemos crear conjuntos de datos, un gráfico que los represente, insertarlo en nuestra aplicación y, gracias al ChartPanel, imprimir, hacer zoom sobre el gráfico, exportarlo a PNG, etcétera. En realidad, no necesitaríamos nada más, pero si queremos cambiar un poco el aspecto de los gráficos, tampoco es necesario realizar excesivas modificaciones, así que veremos algunas de las más comunes.



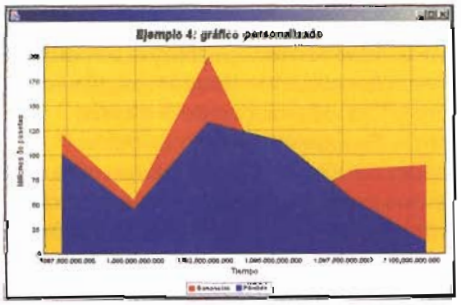
Menú emergente de un ChartPanel.

Colorido del gráfico

Lo primero que podemos modificar es el área externa al gráfico, es decir, el área donde se encuentra el título, la leyenda, los ejes, etc. La clase *JFreeChart* nos ofrece algunos métodos interesantes en este caso, por ejemplo, el método *setBackgroundPaint(Paint paint)* al cual podemos pasar por parámetro un color, y éste será el color que mostrará ese área externa. También tenemos el método *setBorderPaint(Paint paint)* que nos permite, del mismo modo, cambiar el color del borde externo del gráfico. En los ejemplos éste coincide con el borde de la ventana, pero si incrustamos el gráfico en medio de una tabla, por ejemplo, el borde se verá claro.

Por otro lado, podemos modificar el colorido del área del gráfico, esto es, del gráfico en sí. Para ello, tenemos que acceder a dicho área y esto se hace con el método *getPlot()* de la clase JFreeChart, o sea, de nuestro gráfico.

```
Plot miPlot = miGrafico.getPlot();
```



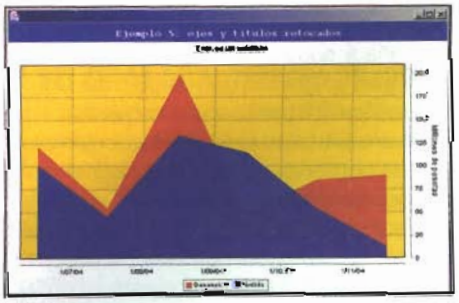
Ejemplo 4. Personalización de los colores.

Y a partir de ahí, la clase *Plot* nos ofrece los métodos necesarios para personalizar el interior del gráfico. Tenemos el método *setBackgroundPaint(Paint paint)* con el que podremos modificar el color de fondo del gráfico, sobre el que se muestran las series. También contamos con el método *setBackgroundAlpha(float alfa)* que nos permitirá seleccionar el nivel de transparencia que deseamos que tenga el color de fondo. El valor pasado por parámetro debe ser un *float* entre 0 y 1, con decimales por tanto. Del mismo modo, el método *setForegroundAlpha(float alfa)* sirve para modificar el nivel de transparencia de las series. Esto es especialmente importante en gráficos de áreas donde un área puede solapar a otra y taparla correctamente. Estableciendo un nivel de transparencia, se verá que detrás del área principal hay otra que está tapada. Y por último, otro método importante de la clase *Plot* es *setOutline Paint(Paint paint)*, que nos permite establecer el color del borde del gráfico.

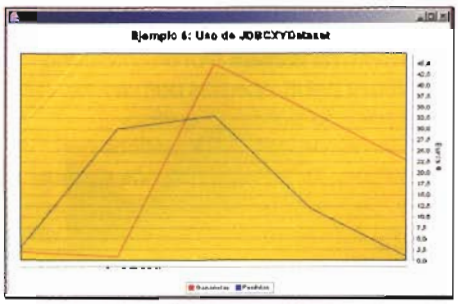
Además de esto, para gráficos que representan XYSeries, podemos acceder a un tipo especial de *Plot*, el *XYPlot* que permite algunas opciones más. Para ello usamos el método *getXYPlot()* de JFreeChart, de nuestro gráfico. Así contaremos con un *XYPlot* que nos permitirá, con respecto a los colores, modificar el color de las líneas verticales y horizontales del gráfico. Para ello usaremos los métodos *setRange GridlinePaint(Paint paint)* y *setDomain GridlinePaint(Paint paint)*, respectivamente. En el fichero *ejemplo4.java* podemos ver el uso de todos estos métodos sobre el *ejemplo3.java*. Existen más posibilidades, pero éstas son las principales.

Los ejes

La clase *XYPlot* nos ofrece también la posibilidad de modificar algunos aspectos de los ejes. Obtenemos el *XYPlot*, de la misma forma explicada en el apartado anterior, y hacemos uso de algunos de sus métodos. El primero es *setAxisRangeLocation (AxisLocation location)* que establece en qué lugar queremos que aparezca el eje del rango. Debemos pasarle como parámetro una de las constantes definidas en la clase *AxisLocation* y permite ponerlo a izquierda o derecha (arriba o abajo si cambiamos de orientación el gráfico). De la misma forma, tenemos el método *setDomainAxisLocation(AxisLocation location)* que nos permite lo mismo pero para el eje de dominios, el eje X. Luego tenemos el método *setAxisOffset(Spacer offset)* que permite definir la distancia entre los ejes y el gráfico en sí, que por defecto están unidos. Usa como parámetro un objeto *Spacer*. Éste requiere como parámetros el tipo de espaciado (absoluto o relativo) y las distancias de izquierda, derecha, arriba o abajo con respecto a los lados del gráfico. Por último, observando la figura siguiente (Ejemplo 5) vemos que, aunque hemos definido el tiempo en meses para el



Ejemplo 5. Personalización de los ejes y títulos.



Ejemplo 6. Uso de JDBCXYDataset.

eje X, se muestra una serie de números que parece no tener fin. Eso es así porque por defecto se muestra en segundos, siguiendo el patrón de UNIX. Para mostrar en meses ese valor temporal, hay que especificarlo. Usamos para ello el método *setDomainAxis(ValueAxis axis)* del objeto *XYPlot*. Este método usa como parámetro un objeto *ValueAxis* (usaremos *DateAxis*, una subclase, en este ejemplo concreto). A su vez la clase *DateAxis* requiere como parámetro en su método *setTickUnit (DateTickUnit unit)* un objeto de tipo *DateTickUnit* que especificará, en este caso, que se trata de meses. Es un poco enrevesado, pero la flexibilidad que ofrece es importante. En el fichero *ejemplo5.java* que se encuentra en el CD se puede ver el uso de estos métodos aplicados sobre el *ejemplo4.java*

Los títulos

Con respecto al título, es un método de la clase *JFreeChart*, o sea, de nuestro gráfico, el que nos permite acceder a sus propiedades. El método en cuestión es *getTitle()* que nos devolverá un objeto *TextTitle* cuyos métodos serán los que utilizaremos. Primero tenemos el método *setBackgroundPaint(Paint paint)* que nos permite cambiar el color de fondo del título Debemos pasarle un objeto, por ejemplo, de la clase *Color*. El método *setPaint(Paint paint)* nos deja seleccionar el color del texto del título, y se usa de igual forma que el anterior.

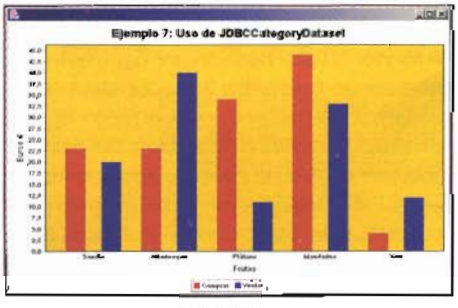
setFont(Font font) sirve para seleccionar el tipo de letra con la cual se mostrará el título. Necesita como parámetro un tipo *Font*. Por último, si además del título queremos añadir subtítulos (títulos de menor orden) en un gráfico, debemos crear un objeto *TextTitle* para cada uno de ellos, crear una lista, tipo *LinkedList*, por ejemplo que los contenga, y por último, usar el método *setSubtitles(List list)* de la clase *JFreeChart*, para añadir estos subtítulos al gráfico. Todos estos métodos se usan en el fichero *ejemplo5.java*, que está en el CD de la revista.

CONTEXTO DE LOS GRÁFICOS

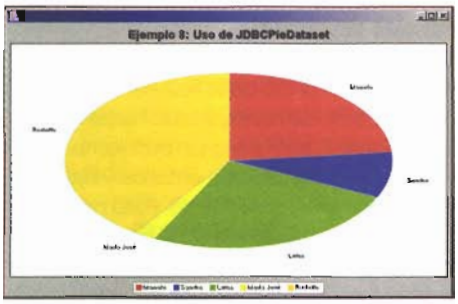
Se entiende como contexto del gráfico el entorno donde va a ser utilizado. Hasta ahora hemos visto ejemplos que se usan para aplicaciones Java, en los que los datos son ofrecidos directamente por la propia aplicación. Esto no está mal, pero la tendencia clara actual, es almacenar los datos en gestores de bases de datos externas e independientes de la aplicación. Sería deseable poder conectar las gráficas directamente a esa base de datos. Por otro lado, para aplicaciones reducidas esto funciona bien, pero para soluciones grandes y distribuidas, por ejemplo, un sistema de información en una intranet, este modelo no vale porque todo se sirve a través de la web. JFreeChart aporta potentes mecanismos para soportar estos requerimientos.

Fuentes de datos en BB.DD.

Para el acceso a bases de datos directamente desde las gráficas, JFreeChart proporciona tres tipos de conjuntos de datos distintos: *JDBCXYDataset*, *JDBCCategoryDataset* y *JDBCPieDataset*. El primero de ellos se utiliza para obtener datos que deben servir para gráficos que necesiten los datos en pares (x,y); el segundo para gráficos que necesiten los datos dados en categorías, y el tercero para la construcción de gráficos de sectores. Todos funcionan de la misma forma. Se crea el conjunto de datos mediante un constructor que necesita ciertos parámetros: la URL de la base de datos a consultar, el controlador JDBC a utilizar, el nombre de usuario y la clave. Una vez que hemos creado el conjunto de datos, hay que realizar una consulta a la base de datos para obtener los datos a representar. Se usa el método *executeQuery(String sql)* que todos estos conjuntos de datos poseen. Como parámetro se pasará la consulta SQL que deseemos. Cada campo que consultemos será una serie del conjunto de datos, pero según el tipo existen algunas restricciones



Ejemplo 7. Uso de JDBCCategoryDataset.



Ejemplo 8. Uso de JDBCPieDataset.

En el tipo *JDBCXYDataset*, la consulta debe devolver solo columnas numéricas y al menos dos de ellas. La primera será el eje de la X del gráfico, la siguiente, la primera serie (eje y). Si aparecen más columnas, todas ellas serán series Si queremos representar ganancias y pérdidas frente al tiempo, la consulta deberá devolver como primera columna el tiempo, y el resto de las columnas serán las ganancias y las pérdidas

JDBCCategoryDataset necesita que la consulta devuelva al menos dos columnas. La primera de ellas, obligatoriamente un texto, que será el nombre de la categoría. La siguiente y posteriores deben ser necesariamente números Así, si queremos mostrar ventas de patatas, ajos y cebollas, la primera columna debe ser el nombre de la verdura y las demás, sus correspondientes valores de ventas.

JDBCPieDataset requiere que la consulta devuelva exclusivamente dos columnas. La primera de ellas será de texto y representará el nombre de cada categoría, y la segunda numérica y representará los valores.

Tras ello, se construyen los gráficos como hemos explicado hasta el momento, el acercamiento a las bases de datos implica solo la carga de datos, pero el resto del proceso no cambia. En los ficheros *ejemplo6.java*, *ejemplo7.java* y *ejemplo8.java* se muestra el uso de estos tres modelos. Para ello he usado una base de datos llamada *miBD* con tres tablas El SQL correspondiente para generar las tablas y los datos de las mismas se encuentra en el CD de la revista en el fichero *SQLEjemplos.zip*.

CONCLUSIONES

Con lo que hemos visto es más que suficiente para introducir con garantías gráficas en la mayoría de las aplicaciones que podamos desarrollar. Con pocas líneas hemos conseguido muy buenos resultados. Con un poco más de investigación personal, el lector conseguirá resultados espectaculares y, lo más importante, de forma libre y gratuita.

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

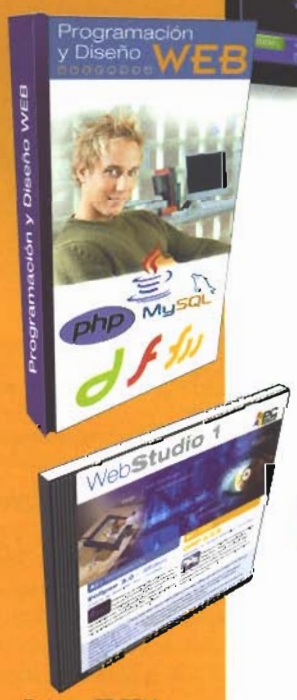
Diseño web
Sección dedicada íntegramente a la suite Macromedia Studio MX 2004
> Dreamweaver MX 2004
> Flash MX 2004
> Fireworks 2004

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