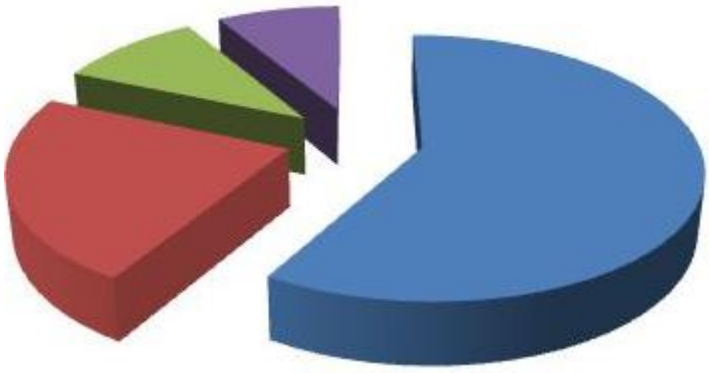
**UNIVERSIDAD DE CÓRDOBA**
**ESCUELA POLITÉCNICA SUPERIOR**
**GRADO EN INGENIERÍA INFORMÁTICA**

**“ SOFTWARE LIBRE Y COMPROMISO SOCIAL”**

**CURSO 2015 ~ 2016**

# ~ JFreeChart ~
# GRÁFICAS EN JAVA



---

**Autor:**

***D***avid ***C***heca ***H***idalgo i32chhid@uco.es

# ÍNDICE:

# ANEXOS:

**ANEXO I:** **FIGURAS**



**ANEXO II:** **TABLAS**

## 1.- INTRODUCCIÓN

Esta práctica es realizada con la finalidad de poder superar la asignatura de Software Libre y Compromiso Social, correspondiente al cuarto curso del Grado en Ingeniería Informática de la Universidad de Córdoba. El encargo de su realización tiene como plazo máximo de entrega el nueve de mayo de dos mil dieciséis y es realizado por parte de D. Domingo Ortiz Boyer, profesor que imparte dicha materia.

Su objetivo principal será la confección de una guía práctica donde se muestren las principales funcionalidades de JFreeChart: librería para la generación de gráficos en Java.

JFreeChart es una biblioteca gráfica libre para Java (TM). Se ha diseñado para su uso en aplicaciones, applets, servlets y JSP. JFreeChart se distribuye con código fuente completo sujeto a los términos de la Licencia Pública General Reducida de GNU, que permite que JFreeChart sea utilizado en propiedad o en aplicaciones de software libre.

JFreeChart puede generar gráficos de sectores, gráficos de barras (regulares y apilados, con la opción de efecto 3D), gráficos de líneas, gráficos de dispersión, gráficos de series de tiempo, diagramas de Gantt, diagramas de medida (línea, brújula y termómetro) y tablas de símbolos, entre otros.

Es conveniente el conocimiento de nociones básicas de programación, al mismo tiempo que se recomienda tener un contacto previo con el lenguaje de programación Java, lo cual facilitará al lector el seguimiento de los ejemplos explicativos que se incluyen en el presente documento.

En su totalidad, la estructura de cada una de las aplicaciones que se incluyen está implementada en Java, empleando para ello el entorno de desarrollo integrado Eclipse.

## 2.- PRERREQUISITOS E INSTALACIÓN

### *¿Cómo descargar las librerías?*

Puede descargar la última versión jfreechart-1.0.1.tar.gz del 27-Enero-2006 en:

http://www.jfree.org/jfreechart/download/

JFreeChart utiliza la biblioteca de clases JCommon (actualmente la versión 1.0.8). El archivo jar de JCommon se incluye en la descarga de JFreeChart, pero si se requiere el código fuente (recomendado) entonces se debe descargar JCommon de:

http://www.jfree.org/jcommon/download/

Después de descargar JFreeChart, es necesario descomprimir los archivos y moverlos a un directorio conveniente. Cuando se desempaquete JFreeChart, se creará en la misma ubicación que el archivo zip o tar.gz un nuevo subdirectorio (jfreechart-1.0.4).

Hay un documento separado en PDF para JCommon, que incluye instrucciones completas para descargar y descomprimir los archivos:

http://www.object-refinery.com/jcommon/index.html

### *¿Cómo instalar lar librerías descargadas anteriormente?*

La instalación será mucho más fácil si hemos colocado las librerías descargadas dentro de nuestro proyecto de Eclipse denominado Graficas (en nuestro caso) Aunque esto no es del todo necesario facilita la tarea a la hora de añadir los .jar.

Las librerías que descargamos (JFreeChart y JCommon) las descomprimiremos en una carpeta llamada lib dentro de nuestro proyecto de Eclipse. Al descomprimir la última versión de ambas librerías obtendremos más archivos que los dos .jar que necesitamos. En este momento, todavía no somos capaces de utilizar las clases que hemos descargado en nuestro proyecto Java. Para poderlas utilizar, las debemos asociar al proyecto, en el Java Build path. Para ello simplemente debemos hacer un click con el botón derecho sobre el proyecto Graficas, y en el menú que nos aparece, seleccionar la opción de Propiedades o Properties. De este modo se nos desplegará el formulario de propiedades que muestra la Figura 1 del Anexo I.

Deberemos seleccionar en el menú izquierdo la opción Java Build Path, y puesto que los ficheros de las librerías se encuentran físicamente dentro del mismo proyecto, deberemos utilizar el botón Add JARs de la pestaña Libraries, el cual a su vez lanza otra ventana donde podemos seleccionar las librerías descargadas. En la carpeta llamada jfreechart-1.0.1 que hemos descomprimido en nuestro proyecto deberemos buscar para añadir el archivo: jfreechart-1.0.1-demo.jar y en la carpeta jcommon-1.0.8 el archivo jcommon-1.0.8.jar. Una vez hecho esto le daremos a aceptar. Si los ficheros estuvieran en una carpeta externa al proyecto deberemos seleccionar el botón Add External JARs y seguir el mismo proceso del caso anterior.

Ahora los archivos de las librerías, que anteriormente teníamos colgando de la carpeta lib, han desaparecido, y se han integrado como un elemento más dentro del propio proyecto Graficas.
Incluso podemos desplegar las librerías y observar los paquetes y las clases que componen la misma. Además, en este caso podremos ver los atributos y métodos

públicos de la clase, lo cual es muy útil para familiarizarnos con el funcionamiento de ambas librerías.

## 3.- FUNDAMENTOS TEÓRICOS

En esta sección trataremos de abordar los conocimientos teóricos que consideramos que serán de gran ayuda y utilidad a la hora de trabajar con JFreeChart.

Para ello incluimos una serie de definiciones que aportarán al lector la información suficiente, así como contribuirán a la comprensión de la presente memoria por muy básico que sea su nivel de conocimientos sobre Java, aunque es conveniente que cuente con nociones sobre programación.

**JFreeChart:** es un marco de software open source para el lenguaje de programación Java, el cual permite la creación de gráficos complejos de forma simple. También trabaja con GNU Classpath, una implementación en software libre de la norma estándar de biblioteca de clases para el lenguaje de programación Java2

**Biblioteca:** es un conjunto de subprogramas utilizados para desarrollar software. Contiene código y datos, que proporcionan servicios a programas independientes. Esto permite que el código y los datos se compartan y puedan modificarse de forma modular. Algunos programas ejecutables pueden ser a la vez programas independientes y bibliotecas, pero la mayoría de estas no son ejecutables. Ejecutables y bibliotecas hacen referencias (llamadas enlaces) entre sí a través de un proceso conocido como enlace, que por lo general es realizado por un software denominado enlazador.

**Open source:** no significa sólo que permite el acceso al código fuente, sino que también tiene que cumplir con los siguientes criterios:

- *Redistribución Libre:* La licencia no debe requerir un canon o cuota por su venta.
- *Código Fuente:* El programa debe incluir el código fuente y debe permitir su distribución.
- *Trabajos Derivados:* La licencia debe permitir modificaciones y trabajos derivados, así como que se distribuyan bajo los mismos términos que la licencia original.
- *Integridad del código fuente del autor:* La licencia puede requerir que trabajos derivados lleven un nombre distinto o número de versión del software original.
- *No discriminación contra personas o grupos:* Debe ser igualmente elegible para contribuir a fuentes abiertas la máxima diversidad de personas y grupos.
- *No discriminación en función de su tratamiento:* La licencia no debe restringir a nadie que haga uso del programa en un campo específico o actividad.
- *Distribución de la licencia:* Los derechos vinculados al programa deben aplicarse a todos aquellos a quienes se redistribuya el programa, sin necesidad de pedir una licencia adicional para estas partes.
- *La licencia no debe ser específica de un producto:* Los derechos vinculados al programa no deben depender de ser parte del programa de distribución de software en particular.
- *La licencia no debe restringir otro software:* La licencia no debe imponer restricciones sobre otro software que es distribuido junto con el software licenciado.
- *La licencia debe ser tecnológicamente neutral:* Ninguna disposición de la licencia puede basarse en una tecnología o estilo de interfaz.

**GNU Classpath**: es un proyecto que propone crear una implementación libre de la biblioteca de clases estándar para el lenguaje de programación Java. Actualmente ya se puede usar *GNU Classpath* para ejecutar programas tan populares como Vuze o Eclipse.
El desarrollo de GNU Classpath está siempre en progreso. Ha habido varias versiones públicas 0.x, las cuales han contribuido hacia el primer gran lanzamiento 1.0.
El desarrollo del código fuente actual se encuentra a través de un servidor anónimo CVS de GNU, y está disponible en:

ftp://ftp.gnu.org/gnu/classpath/

**API:** es una Interfaz de Programación de Aplicaciones (API: por sus siglas en inglés) provista por los creadores del lenguaje Java, y que da a los programadores los medios para desarrollar aplicaciones Java.
Como el lenguaje Java es un Lenguaje Orientado a Objetos, la API de Java provee de un conjunto de clases utilitarias para efectuar toda clase de tareas necesarias dentro de un programa.
La API Java está organizada en paquetes lógicos, donde cada paquete contiene un conjunto de clases relacionadas semánticamente.

**Applet:** Un Java applet es un código JAVA que carece de un método main, por eso se utiliza principalmente para el trabajo de páginas web, ya que es un pequeño programa que es utilizado en una página HTML y representado por una pequeña pantalla gráfica dentro de ésta.
Por otra parte, la diferencia entre una aplicación JAVA y un applet radica en cómo se ejecutan. Para cargar una aplicación JAVA se utiliza el intérprete de JAVA. En cambio, un applet se puede cargar y ejecutar desde cualquier explorador que soporte JAVA (Internet Explorer, Mozilla Firefox, Google Chrome, Netscape...).

**Servlet:** La palabra servlet deriva de applet. Por contraposición, un servlet es un programa que se ejecuta en un servidor. El uso más común de los servlets es generar todas páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web.

# 4.- GUÍA DE USO PARA EL PROGRAMADOR

En este apartado explicaremos cómo crear gráficos de diferentes tipos con la librería de código abierto de Java JFreeChart con unos pequeños códigos ejemplo que podemos encontrar en la propia web de los creadores de la librería o en su guía oficial del desarrollador.

Todos los métodos explicados en este apartado han sido consultados del JavaDoc oficial de JFreeChart citado en el apartado de documentación o en el JavaDoc de Java.

## 4.1.- Dibujar un gráfico con JFreeChart

Crear gráficas con JFreeChart, se compone de tres pasos totalmente necesarios y muy simples:

1. Crear un dataset que contenga los datos a mostrar en las gráficas.
2. Crear una instancia de la clase JFreeChart que será la responsable de dibujar las gráficas.
3. Dibujar/Imprimir la gráfica sobre algún objeto que lo permita (a menudo, pero no siempre, un panel por pantalla);

## 4.2.- Gráficos circulares o pastel

A continuación, se detalla una pequeña explicación sobre cómo crear nuestro primer gráfico circular o pastel (como es denominado en la librería de JFreeChart) con los tres pasos muy simples citados anteriormente:

Crearemos un proyecto nuevo en Eclipse, este puede llamarse como deseemos (Gráficas en nuestro caso), y una clase llamada GraficoCircular (nombre opcional, pero deberemos respetarlo en el interior de la clase).

Comenzamos por importar las librerías que vamos a necesitar en este pequeño ejemplo dentro de nuestra clase GraficoCircular:

```
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.data.general.DefaultPieDataset;
```

A continuación, comenzamos con el paso uno de los tres simples pasos citados, crearemos el dataset. El dataset no es más que el conjunto de datos que queremos que se muestre en nuestro gráfico (en este primer ejemplo, nuestro gráfico circular o tarta). Para ello se declara un nuevo objeto de tipo DefaultPieDataset que nos servirá para establecer los datos que mostrará el gráfico:

```
DefaultPieDataset dataset = new DefaultPieDataset();
```

Después definiremos una serie de categorías (en el caso del ejemplo serán tan solo tres) mediante la llamada al método setValue cuyos parámetros son:

```
void setValue(java.lang.Comparable key, double value);
```

donde:

- **key** es el nombre entre “ “ o como cadena de caracteres que queramos darle a esa categoría.
- **value** es el valor que representa dentro del gráfico circular esa categoría refiriéndonos al tanto por ciento (%) formato double o convertible a éste.

Las llamadas a dataset.setValue serían como sigue:

```
dataset.setValue("Categoria 1", 20.5);
dataset.setValue("Categoria 2", 60.7);
dataset.setValue("Categoria 3", 18.8);
```

Una vez definidos los datos a mostrar en el paso dos deberemos definir cómo vamos a presentar el conjunto de datos creado en el apartado anterior. Tenemos que crear un objeto de tipo JFreeChart que pueda dibujar un gráfico con los datos de nuestro gráfico circular. Para ello vamos a utilizar la clase ChartFactory. Es una clase cuyos métodos son estáticos de tal manera que no necesitaremos crear un objeto de este tipo para usar los métodos que implementa. El método que necesitaremos para un gráfico circular es el siguiente:

```
public static JFreeChart createPieChart(java.lang.String title,
        PieDataset dataset,
        boolean legend,
        boolean tooltips,
        boolean urls);
```

Parámetros:

- **title** es el título del gráfico (null permitido).
- **dataset** es el conjunto de datos para el gráfico (null permitido).
- **legend** indicamos si deseamos o no leyenda en el gráfico.
- **tooltips** sirve para configurar la gráfica y generar sugerencias de la herramienta.
- **URL** configura la gráfica para generar direcciones URL.

Devuelve:

- Un gráfico circular.

La llamada en nuestro caso sería como sigue:

```
JFreeChart chart = ChartFactory.createPieChart(
        "Grafico circular",
        dataset,
        true,
        true,
        false);
```

Tan sólo nos queda el último de los tres pasos: mostrar/imprimir por pantalla el gráfico resultante. En nuestro caso es una impresión por pantalla, pero JFreeChart permite además otros tipos de salidas gracias a su uso de la clase Graphics2D. Para ello necesitaremos declarar un objeto de la clase ChartFrame. Usaremos en su creación el primero de los dos constructores que implementa:

```
ChartFrame(java.lang.String title, JFreeChart chart);
```

La llamada en nuestro caso sería:

*ChartFrame frame = new ChartFrame("Test", chart);*

El resto de funciones necesarias proceden de clases de java de las que hereda ChartFrame como es java.awt.Window de la que hereda:

- El método void pack(); cuya función es redimensionar la ventana para que tenga un tamaño que se ajuste al tamaño preferido y al diseño de sus subcomponentes.
- El método void setVisible(boolean b); cuya función es mostrar u ocultar la ventana en función del valor del parámetro b.

Las llamadas en nuestro caso son:

*frame.pack();*
*frame.setVisible(true);*

Una vez explicado el proceso detenidamente para familiarizarnos con la librería y su uso, puede consultarse el código completo del ejemplo en el Anexo III.

Ya tenemos el programa completo así que teniendo seleccionado el fichero en el que se encuentra el código, pulsaremos el botón de play de la barra superior del menú que ofrece Eclipse y se abrirá una nueva ventana con nuestro gráfico, tal y como muestra la Figura II del Anexo I.

Las escalas de los ejes y leyendas serán incluidas automáticamente. Pulsando el botón derecho del ratón sobre el gráfico, se puede acceder a las propiedades del mismo, hacer zoom en la interfaz del gráfico, cambiar algunos ajustes a través del menú local, así como guardarlo como archivo png.

### 4.3.- Gráficos de barras

En este apartado se ofrece una introducción a la creación de gráficos de barras con JFreeChart. Para ello usaremos un gráfico de barras muy sencillo e iremos explicando cada nuevo método o funcionalidad que encontremos.
Los gráficos de barras o columnas son usados para comparar dos o más valores que pueden estar orientados horizontal o verticalmente.

En JFreeChart este cuadro se conoce como un conjunto de datos o *dataset*, cada encabezado de la columna es una categoría, y cada fila de la tabla es una serie (Véase la Tabla 1 incluida en el Anexo II)

Lo primero que debemos hacer es colocar los import que necesitaremos para las clases y métodos usados más adelante:

*import org.jfree.chart.ChartFactory;*
*import org.jfree.chart.ChartFrame;*
*import org.jfree.chart.JFreeChart;*
*import org.jfree.chart.plot.PlotOrientation;*
*import org.jfree.data.category.DefaultCategoryDataset;*

Crearemos una clase llamada EjemploBarras dentro de la cual implementaremos el gráfico de barras.

Para crear un gráfico de barras nos sustentaremos en una clase específica de JFreeChart del mismo modo que hicimos el grafico circular en el anterior apartado. En este caso la clase que utilizaremos es: *DefaultCategoryDataset.*

Crearemos una nueva variable del tipo DefaultCategoryDataset y reservaremos memoria para ésta:

```
DefaultCategoryDataset dataset = new DefaultCategoryDataset();
```

Definiremos el dataset o conjuntos de datos que serán representados en el gráfico de barras, para ello usaremos el método addValue() similar al anterior setValue().

Los parámetros son:

void addValue(double value, java.lang.Comparable rowKey, java.lang.Comparable columnKey);

Donde:

- **value** es el valor de tipo double que se le asignará a una componente del gráfico.
- **rowKey** es la clave identificadora de una fila para más adelante acceder a ésta.
- **columnKey** es la clave identificadora de una columna para más adelante acceder a ésta.

Elegiremos ésta en lugar de setValue() porque addValue() permite añadir directamente un float en lugar del objeto Number como dato del gráfico.

Mirando la Tabla 1 del Anexo II comenzaremos a añadir los datos a nuestro gráfico:

```
dataset.addValue(2.0, "Fila 1", "Columna 1");
dataset.addValue(3.0, "Fila 1", "Columna 2");
dataset.addValue(5.2, "Fila 1", "Columna 3");
dataset.addValue(1.7, "Fila 2", "Columna 1");
dataset.addValue(2.4, "Fila 2", "Columna 2");
dataset.addValue(2.8, "Fila 2", "Columna 3");
```

El siguiente paso es crear una instancia de JFreeChart que dibuja un gráfico de barras para este conjunto de datos o *dataset*.
Para esto necesitaremos dos clases: JFreeChart y ChartFactory como en el gráfico del anterior apartado. En este caso utilizaremos el método estático createBarChart de la clase ChartFactory cuyos parámetros son:

```
static JFreeChart createBarChart(java.lang.String title, java.lang.String
categoryAxisLabel, java.lang.String valueAxisLabel, CategoryDataset dataset,
PlotOrientation orientation, boolean legend, boolean tooltips, boolean urls);
```

Donde:

- **title** es el título que tendrá nuestro gráfico.
- **categoryAxisLabel** es la leyenda que poseerá el eje x.
- **valueAxisLabel** es la leyenda que poseerá el eje y.
- **dataset** el conjunto de datos que se representarán en el gráfico.

- **orientation** aquí indicamos si las columnas estarán verticales u horizontales mediante: *PlotOrientation.VERTICAL y PlotOrientation.HORIZONTAL.*
- **legend** lo marcamos como true si deseamos leyenda y como false en caso contrario.
- **tooltips** lo marcamos como true si queremos ver información sobre herramientas cuando se muestra el gráfico
- **urls** si queremos crear urls

```
JFreeChart chart = ChartFactory.createBarChart(
        "Primer grafico de barras.",
        "Datos",
        "Valor",
        dataset,
        PlotOrientation.VERTICAL,
        true,
        true,
        false
);
```

Por ultimo sólo nos queda crear un objeto de tipo ChartFrame que será el que más tarde imprimiremos por pantalla tal y como vimos en el apartado anterior:

```
ChartFrame frame = new ChartFrame("Primer gráfico de barras.",chart);
```

Ahora mediante métodos en la librería java.awt.Window definiremos el tamaño que tendrá nuestra ventana ( la cual contiene al gráfico ) y la activaremos para mostrarla:

```
frame.pack();
frame.setVisible(true);
```

Una vez explicado el proceso detenidamente para familiarizarnos con la librería y su uso, puede consultarse el código completo del ejemplo en el Anexo IV y una muestra del gráfico obtenido como resultado, consultando la Figura 3 del Anexo I.

### 4.4.- Gráficos de líneas

En este caso, vamos a explicar paso a paso como poder programar nuestro primer gráfico de líneas, siguiendo los tres pasos sencillos que son necesarios para la creación de cualquier gráfico con la librería que estamos usando, JFreeChart.

En primer lugar, podemos crear un nuevo proyecto eclipse o en un proyecto existente, insertaremos una nueva clase llamada GraficoLineas, por ejemplo. Este nombre podrá seleccionarlo el programador, aunque tendrá que ser tenido en cuenta en el desarrollo interior de la clase.

Comenzamos por importar las librerías que vamos a necesitar en este pequeño ejemplo dentro de nuestra clase GraficoLineas:

```
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.data.category.DefaultCategoryDataset;
```

```
import org.jfree.chart.plot.PlotOrientation;
```

Continuamos llevando a cabo el primero de los tres simples pasos citados anteriormente, es decir, la creación del dataset que será el conjunto de datos que serán mostrados en nuestro gráfico de líneas. En este caso se declarará un nuevo objeto de tipo DefaultCategoryDataset.

```
DefaultCategoryDataset dataset = new DefaultCategoryDataset();
```

Seguidamente definimos las categorías (cinco para el ejemplo) a través del método addValue cuyos parámetros son:

```
void addValue(double value, java.lang.Comparable rowKey,
java.lang.Comparable columnKey);
```

Donde:

- **value** es el valor de tipo double que se le asignará a una componente del gráfico.
- **rowKey** es la clave identificadora de una fila para más adelante acceder a ésta.
- **columnKey** es la clave identificadora de una columna para más adelante acceder a ésta.

En nuestro ejemplo, se corresponde con las siguientes sentencias:

```
dataset.addValue(212, "Clave Leyenda", "Columna 1");
dataset.addValue(504, "Clave Leyenda", "Columna 2");
dataset.addValue(1520, "Clave Leyenda", "Columna 3");
dataset.addValue(1842, "Clave Leyenda", "Columna 4");
dataset.addValue(2991, "Clave Leyenda", "Columna 5");
```

Como rowKey indicamos siempre el mismo parámetro para que el gráfico muestre una línea que una todos los puntos representados, ya que en caso contrario, sólo podríamos observar los citados puntos correspondientes a los valores insertados en el dataset. Al mismo tiempo, éste será el valor que se muestre en la leyenda.

Realizado el paso anterior, pasamos al siguiente y procedemos a la creación de un objeto de la clase estática ChartFactory mediante la llamada a su siguiente método:

```
static JFreeChart createLineChart(java.lang.String title, java.lang.String
categoryAxisLabel, java.lang.String valueAxisLabel, CategoryDataset dataset,
PlotOrientation orientation, boolean legend, boolean tooltips, boolean urls);
```

Donde:

- **title** es el título que tendrá nuestro gráfico.
- **categoryAxisLabel** es la leyenda que poseerá el eje x.
- **valueAxisLabel** es la leyenda que poseerá el eje y.
- **dataset** el conjunto de datos que se representarán en el gráfico.
- **orientation** aquí indicamos si las columnas se considerarán verticales u horizontales mediante: *PlotOrientation.VERTICAL y PlotOrientation.HORIZONTAL.*

- **legend** lo marcamos como true si deseamos leyenda y como false en caso contrario.
- **tooltips** lo marcamos como true si queremos ver información sobre herramientas cuando se muestra el gráfico.
- **urls** si queremos crear urls.

Devuelve:

- Un gráfico de líneas.

Las sentencias correspondientes a nuestro ejemplo son las siguientes:

```
JFreeChart chart = ChartFactory.createLineChart(
"Grafico de Lineas",
"Eje de la x",
"Eje de la y",
dataset,
PlotOrientation.VERTICAL,
true,
true,
false);
```

En último lugar, tendremos que crear un objeto del tipo ChartFrame, el cual emplearemos para imprimir por pantalla nuestro gráfico:

```
ChartFrame frame = new ChartFrame("Prueba", chart);
```

Ahora mediante métodos de la librería java.awt.Window podremos activar la ventana que contendrá el gráfico, así como mostrarla :

```
frame.pack();
frame.setVisible(true);
```

La exposición del código completo de nuestro ejemplo está disponible en el Anexo V, así como el gráfico resultante en la Figura IV del Anexo I.

## 4.5.- Funciones de personalización de los gráficos

En este capítulo mostraremos algunas funciones que puede aplicarse a los gráficos anteriormente descritos para su personalización como puede ser el cambio del color de fondo, tipo de fuente, etc.

### ♦ Personalización de un gráfico circular.

Comenzaremos con la clase PiePlot y algunas de sus funciones, las cuales pueden ser consultadas en el propio JavaDoc de JFree Chart.
Esta clase hereda de la clase Plot, y es la específica para gráficos circulares o de tarta.

Lo primero que deberemos hacer para comenzar a usar sus funcionalidades es crear una instancia de la clase referenciando el gráfico sobre el que haremos la personalización mediante el uso del objeto JFreeChart chart que deberemos haber creado con anterioridad y que era necesario en nuestro programa:

```
PiePlot plot = (PiePlot) chart.getPlot();
```

Una vez hecho esto ya podemos comenzar a explorar más a fondo sus funcionalidades.

***Clase PiePlot:***

void **setSectionPaint**(java.lang.Comparable key, java.awt.Paint paint);

- **key** hace referencia a la clave con la que denominamos a una de las porciones de nuestro gráfico circular, es el identificador por el que accedemos a una porción para modificar su color.
- **paint** define el nuevo color para esa porción.

Para definir el color recurrimos a la clase Color y sus métodos estáticos para definir el color que deseamos en la sección especificada por key. Veamos un ejemplo:

```
DefaultPieDataset dataset = new DefaultPieDataset();
dataset.setValue("Categoria 1", 20.5);
dataset.setValue("Categoria 2", 60.7);
dataset.setValue("Categoria 3", 18.8);
JFreeChart chart = ChartFactory.createPieChart( "Grafico circular", dataset, true,
true, false);
PiePlot plot = (PiePlot) chart.getPlot();
plot.setSectionPaint("Categoria 1", Color.orange);
```

Con esto conseguimos que la sección definida por "Categoria 1" cambie su color de su color por defecto a naranja, tal y como muestra la Figura 5 del Anexo I.

El resto de funciones que se explican se colocarían de la misma forma que ésta, antes de su impresión por pantalla con estas tres líneas de código:

```
ChartFrame frame = new ChartFrame("Test", chart);
frame.pack();
frame.setVisible(true);
```

Continuamos con más funciones de personalización.

void **setSectionOutlinesVisible**(boolean visible);

- **visible** indica la visibilidad (true) o no (false) de la línea que rodea el círculo del gráfico.

```
plot.setSectionOutlinesVisible(true);
plot.setSectionOutlinesVisible(false);
```

void **setBaseSectionOutlinePaint**(java.awt.Paint paint);

- **paint** indica el color que tendrá la línea exterior del círculo, por ejemplo, rosa, tal y como muestra la Figura 6 del Anexo I.

```
plot.setBaseSectionOutlinePaint(Color.pink);
```

void **setBaseSectionOutlineStroke**(java.awt.Stroke stroke);

- **stroke** define mediante un float el grueso de la línea

*plot.setBaseSectionOutlineStroke(new BasicStroke(2.0f));*

En ambos casos podemos aplicar el color o engrosar la línea sólo de una de las secciones añadiendo un parámetro a cada función:

*void setSectionOutlinePaint(Comparable key, Paint paint);*
*void setSectionOutlineStroke(Comparable key, Stroke stroke);*

void **setExplodePercent**(java.lang.Comparable key, double percent);

- **key** define la porcion que modificaremos
- **percent** el % correspondiente a la porción anterior

```
DefaultPieDataset dataset = new DefaultPieDataset();
dataset.setValue("Categoria 1", 20.5);
dataset.setValue("Categoria 2", 60.7);
dataset.setValue("Categoria 3", 18.8);
JFreeChart chart = ChartFactory.createPieChart( "Grafico circular", dataset, true,
true, false);
PiePlot plot = (PiePlot) chart.getPlot();
plot.setExplodePercent("Categoria 1", 0.205);
```

Debemos tener en cuenta no equivocarnos con el % que le corresponde a Categoria 1, y debemos expresarlo dividiendolo por 100. Si al colocar los valores pusimos *dataset.setValue("Categoria 1", **20.5**);* ahora deberemos indicar *plot.setExplodePercent("Categoria 1", **0.205**);*

Tambien podría indicarse así:

```
plot.setExplodePercent("Categoria 1",
dataset.getValue("Categoria 1").doubleValue()/100);
```

Obteniendo el valor con dataset.getValue("Categoria 1") donde “Categoria 1” es cualquier key, como lo que devuelve es de tipo Number deberemos pasarlo a double con doubleValue() y por último, como indicamos antes, dividido por 100. Esta forma puede parecer más larga pero no cometeremos errores al introducir el valor de la categoría.

Por último mostraremos cómo crear el gráfico en 3D. Es muy simple, tan solo deberemos crear un objeto del tipo JFreeChart pero en lugar de usar el metodo estático createPieChart de la clase ChartFactory, usaremos createPieChart3D();

Existen algunas limitaciones con esta clase:

- Las secciones ampliadas no son compatibles, es decir, no se puede aplicar la función setExplodePercent.
- No es posible ajustar el ángulo de la "rotación" para el efecto 3D en ciertos casos.

(Véase ejemplo de Gráfico Circular 3D en Figura VII del Anexo I)

Una vez creado el gráfico como 3D podemos usar la clase PiePlot3D en lugar de la anterior PiePlot y seguir explorando nuevas formas de personalización, como por ejemplo, cambiar la transparencia del gráfico con:

```
PiePlot3D plot = (PiePlot3D) chart.getPlot();
plot.setForegroundAlpha(0.5f);
```

(Véase Figura VIII del Anexo I)

### ♦ Personalización de un gráfico de barras.

Para personalizar el gráfico de barras usaremos varias clases diferentes.

Podemos modificar el mismo objeto de la clase JFreeChart chart que creamos al definir el gráfico, para ello podemos usar diferentes métodos pertenecientes a esta clase.

Comenzaremos con modificar el fondo, para ello hay varias funciones:

- void **setBackgroundPaint**(java.awt.Paint paint); en la cual indicamos mediante paint y la clase Color el nuevo color que queramos que posea el fondo el gráfico.
- void **setBackgroundImage**(java.awt.Image image); en la cual indicaremos mediante image la imagen que queremos colocar como fondo del gráfico.
- Para indicarle la imagen creamos un objeto image para almacenar una imagen y como parámetros de new File() indicaremos entre " " la ruta completa hasta nuestra imagen, la cual colocaremos dentro de una carpeta en nuestro proyecto, tal y como se muestra a continuación:

  ```
  BufferedImage img = ImageIO.read(new File("src/img/camaleon.jpg"));
  ```

Una vez hecho esto ya podemos hacer la llamada normalmente:

```
chart.setBackgroundImage(img);
```

El código quedaría de esta forma:

```
BufferedImage img = null;
try {
        img = ImageIO.read(new File("imagenes/fractal.jpg"));
} catch (IOException e) {
        e.printStackTrace();
}
chart.setBackgroundImage(img);
```

El resultado aplicando esta imagen sería tal y como muestran las Figuras 9 y 10 del Anexo I.

Como podemos comprobar el resultado no es el óptimo, esto es por haber aplicado el método al objeto chart de tipo JFreeCHart en lugar de aplicarselo a CategoryPlot plot. De esta segunda forma el resultado sería el que muestra la Figura 11 del Anexo I.

**Cabe añadir que estas dos funciones de personalización del fondo del gráfico son aplicables al resto de los gráficos y no sólo al de barras.**

Además, ambas funciones son comunes tanto para objetos de la clase JFreeChart como de la clase Plot:

```
plot.setBackgroundPaint(Color.lightGray); // chart.setBackgroundPaint(Color.lightGray);
plot.setRangeGridlinePaint(Color.white);   // chart.setRangeGridlinePaint(Color.white);
```

Para customizar los colores de las barras y su espaciado deberemos añadir una nueva clase llamada BarRenderer:

- void **setSeriesPaint**(int series, java.awt.Paint paint) nos permite cambiar los colores a las barras. Mediante series indicaremos la fila (orden dentro de las barras de una columna) que queremos cambiar y paint el color deseado.

```
BarRenderer renderer = (BarRenderer) plot.getRenderer();
renderer.setSeriesPaint(0, Color.gray);
```

Nótese que el método setSeriesPaint () se define en la clase base AbstractRenderer y **se puede utilizar para todos los tipos de gráficos**.

- void **setDrawBarOutline**(boolean draw); define mediante una variable booleana draw si contornearemos exteriormente las barras de nuestro grafico (colocándola a true) o no (colocándola a false).

```
renderer.setDrawBarOutline(true); // renderer.setDrawBarOutline(false);
```

- void **setItemMargin**(double percent); permite establecer la separación entre las barras pertenecientes a un mismo grupo de columnas mediante una variable double percent.

```
renderer.setItemMargin(0.0);
```

Podremos ver en conjunto las dos funciones anteriores consultando la Figura 12 del Anexo I.

### ♦ Personalización de un gráfico de barras.

En este caso, vamos a exponer una serie de métodos que nos permitirán personalizar el gráfico de líneas creado anteriormente.

En primer lugar, procederemos a indicar los import que serán necesarios incluir en nuestra clase de Java GraficoLineas para poder hacer uso de ellos, ya que aunque utilizaremos clases estáticas que no necesitan ser instanciadas para poder usar sus métodos, es necesario y muy importante no olvidar referenciarlas.

```
import java.awt.Font;
import org.jfree.chart.title.TextTitle;
```

```
import org.jfree.ui.HorizontalAlignment;
import org.jfree.ui.RectangleEdge;
```

Una vez incluidos, vamos a ir explicando uno a uno los métodos empleados para la personalización del gráfico de líneas que no han sido incluidos en apartados anteriores, por lo que se recomienda consultarlos, en caso de estar interesado en conocer más formas de personalización, ya que existen métodos aplicables a cualquier tipo de gráfico.

Con nuestra instancia de la clase del gráfico de líneas creada, podemos realizar llamadas al siguiente método propio:

- void **addSubtitle**(new TextTitle("Subtitulo")); permite añadir un subtítulo al gráfico.

```
chart.addSubtitle(new TextTitle(“Esto es un ejemplo de subtítulo”));
```

Otra forma de hacerlo o en caso de necesitar incluir texto adicional, como por ejemplo para indicar el nombre de los autores, se puede instanciar la clase TextTitle tal y como se muestra:

- TextTitle source = new TextTitle("Autores: Laura Gata Moreno / David Checa Hidalgo ");

Una vez creado el objeto anterior, podemos configurar algunos de sus atributos a nuestro antojo, como puede comprobarse a continuación:

- void **setFont**(new Font(java.awt.Font, Font.Style, Size)); permite establecer la fuente que tendrá el texto, así como el estilo y el tamaño respectivamente.

```
source.setFont(new Font("TimesNewRoman", Font.ITALIC, 12));
```

- void **setPosition**(org.jfree.ui.RectangleEdge position); permite situar el texto en la parte deseada de la ventana que mostrará el gráfico.

```
source.setPosition(RectangleEdge.BOTTOM);
```

- void **setHorizontalAlignment**(org.jfree.ui.HorizontalAlignment alignment); permite establecer la alineación del texto.

```
source.setHorizontalAlignment(HorizontalAlignment.RIGHT);
```

Una vez configurado el nuevo texto, lo insertamos en el gráfico mediante la siguiente sentencia:

```
chart.addSubtitle(source);
```

Este pequeño ejemplo, insertará un nuevo texto en la parte inferior derecha de la ventana, con el nombre de los autores, alineado a la derecha y empleando la fuente “Times New Roman”, cursiva y de tamaño 12, tal y como puede comprobarse consultando la Figura 13 del Anexo I.

### 4.6.- Aplicación demostrativa

Una vez estudiada y comprendido el funcionamiento de la librería para la generación de gráficos en Java JFreeChart, hemos procedido a la implementación de una sencilla aplicación interactiva que ofrezca al usuario la posibilidad de generar los gráficos básicos explicados detenidamente con anterioridad.

Para ello se ha empleado la notación húngara y las especificaciones oportunas siguiendo los patrones establecidos en JavaDoc.

En primer lugar se ha desarrollado la interfaz IGrafico para las clases de gráficos creadas para este trabajo y que define así una convención de nombrado.
Está compuesta por cuatro métodos que tendrán que ser desarrollados por todas las clases de gráficos que la implementen. No se trata nada más que los métodos correspondientes para poder llevar a cabo cada uno de los tres pasos sencillos que comentamos con anterioridad y que son necesarios para la creación de cualquier gráfico: crear un dataset (diferenciando si se le asigna nombre a las categorías del gráfico o no), crear una instancia de JFreeChart y mostrar/imprimir el gráfico por pantalla.
La interfaz simplemente marca las directrices que tendrán que seguir los métodos de la clase de cada uno de los gráficos, pero tendrá que ser en la clase de cada gráfico donde se implementen. Los métodos de la interfaz son los siguientes:

- public JFreeChart **createChart**();

Crea el objeto JFreeChart encargado de representar el gráfico por pantalla y devuelve el objeto correctamente inicializado.

- public ChartFrame **createFrame(**JFreeChart jfcChart);

Crea el frame o ventana donde se imprime el gráfico, devolviendo el frame o ventana correctamente inicializado.

- public void **defineValues**();

Define los valores del dataset del gráfico correspondiente.

- public void **defineValuesAndNames**();

Define los valores del dataset y los nombres de las categorías del gráfico correspondiente.

Puede consultarse el código completo de la citada interfaz en el Anexo VI.

A su vez, se ha implementado la clase Errors para el control de errores en la introducción de parámetros por teclado.

Contiene siete métodos estáticos, es decir, no es necesaria la existencia de una instancia a dicha clase para poder hacer uso de ellos, los cuales son explicados a continuación:

- static public boolean **yesNoError**(String strName, String strMessage)

Controla que el contenido de strName sea sí o no sin atender a mayúsculas o minúsculas y solicita la introducción de un dato mientras que no se teclee alguno de los valores anteriores. El parámetro recibido strMessage es empleado para repetir, en caso de error, el mismo mensaje que se ha mostrado por pantalla realizando la solicitud. Una vez introducido el valor esperado, lo devuelve contenido en strName.

- static public int **readInteger**(String strMessage)

Controla que el dato leído por teclado sea un valor entero, solicitándolo de forma iterativa en caso contrario. La finalidad del parámetro strMessage es la misma que en método anterior. Devuelve un entero una vez que ha sido introducido correctamente.

- static public int **readPositiveInteger**(String strMessage)

Su funcionalidad es exactamente igual al método anterior, pero en este caso el valor entero deberá ser positivo.

- static public double **readDouble**(String strMessage)

Controla que el dato leído por teclado sea un valor real, solicitándolo de forma iterativa en caso contrario. La finalidad del parámetro strMessage es la misma que en la expuesta en métodos anteriores. Devuelve un valor real una vez que ha sido introducido correctamente.

- static public double **readPositiveDouble**(String strMessage)

Su funcionalidad es exactamente igual al método anterior, pero en este caso el valor real deberá ser positivo.

- static public void **errorPieValues**(Double[] dValues,int iNumber)

Método que comprueba que los valores introducidos en % para las categorías del gráfico circular suman 100 % en total. Si esto no se cumple, se vuelve a solicitar de nuevo su introducción, finalizando el método con un dataset para el gráfico inicializado correctamente.

- static public String **barsHorientation**(String strName, String strMessage)

Método que comprueba si se ha introducido correctamente la orientación (horizontal o vertical), tanto para el gráfico de barras como el de líneas, sin atender a mayúsculas ni minúsculas. Solicita la introducción de un dato mientas que no se ajuste a ninguno de los dos mencionados. La funcionalidad de strMessage sigue siendo la misma que la expuesta en los métodos anteriores y devuelve la cadena “horizontal” o “vertical” contenida en strName.

Puede consultarse el código completo de la clase Errors en el Anexo VII.

Han sido implementadas tres clases, una para cada uno de los gráficos básicos detallados en esta memoria (circular, de barras y de líneas) las cuales definen el gráfico correspondiente a la vez que implementan los métodos necesarios para su tratamiento y que al mismo tiempo vienen definidos por la interfaz IGrafico explicada con anterioridad.

Con la intención de no redundar en información ya ofrecida, se le ofrece al lector la oportunidad de consultar el código completo de dichas clases en los Anexos del VIII al X para los gráficos circular, de barras y de líneas respectivamente.

Para finalizar con la aplicación demostrativa, se ha implementado la clase Prueba que permite interactuar con el usuario a través de la consola.
Comienza solicitando que se introduzca el tipo de gráfico que se quiere crear y una vez introducido correctamente de entre las opciones que ofrece (barras, circular o líneas) solicita que se introduzcan los parámetros necesarios en función del gráfico seleccionado. Esta clase implementa el método main del programa al mismo tiempo que una serie de métodos estáticos que se detallan a continuación:

- static public void **startGraphicPie**(PieGraphic pgGrafico)

Realiza el proceso completo de creación de un gráfico circular así como el control de errores al introducir los parámetros de entrada solicitados. Devuelve un objeto de tipo PieGraphic.

- static public void **startGraphicBars**(BarsGraphic bgGrafico)

Realiza el proceso completo de creación de un gráfico de barras así como el control de errores al introducir los parámetros de entrada solicitados. Devuelve un objeto de tipo BarsGraphic.

- static public void **startLineGraphic**(LineGraphic lgGrafico)

Realiza el proceso completo de creación de un gráfico de líneas así como el control de errores al introducir los parámetros de entrada solicitados. Devuelve un objeto de tipo LineGraphic.

- public static void **main**(String[] strArgs)

Main de la clase de prueba que define los gráficos según lo desee el usuario. No requiere que se le pase ningún argumento desde la línea de comandos de la consola en su invocación.

Al igual que en casos anteriores, se puede consultar el código completo de la clase Prueba en el Anexo XI.

## 5.- ALTERNATIVAS

En esta sección, trataremos de dar a conocer alternativas tanto de código abierto como comercial a la librería para la implementación de gráficos en Java JFreeChart, la cual es motivo de estudio del presente trabajo.

**♦ Código abierto:**

**JCCKit (Java Chart Construction Kit):** El JCCKit es una pequeña biblioteca de código abierto y un marco muy flexible para la creación de gráficos y diagramas científicos. JCCKit está escrito para la plataforma JDK 1.1.8 (a excepción de un procesador de Graphics2D). Por lo tanto, es adecuado para Applets científicos y para ejecutar una aplicación Personal Java en una PDA.

Si desea más información acerca de JCCKit, consulte la Tabla 2 incluida en el Anexo II.

**QN Plot:** Proporciona gráficos de una o más funciones como los componentes Swing. Se diseñó para representar grandes cantidades de datos en tiempo real y requiere Java 5 o superior.

(Véase Tabla 3 incluida en Anexo II)

**JOpenChart:** es una librería de herramientas de Java para insertar gráficos en diferentes tipos de aplicaciones sin importar sin son del tipo Servlets, de escritorio o webs. Hasta su aparición, no existían soluciones satisfactorias de código abierto disponibles, mientras que las comerciales tenían un precio bastante elevado.

(Véase Tabla 4 incluida en Anexo II)

**Openchart2:** basada en la librería JopenChart original de Sebastian Müller ofrece una interfaz para crear gráficos en dos dimensiones, sencilla a la vez que potente, para programadores de Java. Cuenta con un surtido de gráficos, incluyendo gráficos de dispersión avanzada, gráficos de barras y circulares y constantemente le son añadidas nuevas características para mejorar el poder de la librería.

(Véase Tabla 5 incluida en Anexo II)

**PtPlot:** Conjunto de componentes para la creación de gráficos de dos dimensiones escritos en Java.

(Véase Tabla 6 incluida en Anexo II)

**JCharts:** es una utilidad 100% Java basada en gráficos que genera una gran variedad de ellos. El paquete ha sido diseñado desde cero por voluntarios para la visualización de gráficos a través de Servlets, JSP y aplicaciones Swing.
Su primera versión data de diciembre de 2000.

**JChart2D:** es una biblioteca minimalista de gráficos en tiempo real publicada bajo la licencia aprobada OSI GNU Lesser General Public.
Está diseñada para la visualización de múltiples puntos de seguimiento y se centra únicamente alrededor de un widget configurable: Chart2D, el cual se trata

de un JComponent que se puede añadir a una interfaz de usuario Java Swing, por tanto, el conocimiento básico de Java AWT y Swing son importantes.

JChart2D está destinado a tareas de ingeniería y no para las presentaciones. Su especialidad es el tiempo de ejecución, debido a que la visualización dinámica precisa de los datos con una sobrecarga mínima de configuración. Todo el mundo puede utilizar JChart2D de forma gratuita.

**ThunderGraph:** es otra biblioteca de gráficos Java cuya intención es la de dibujar gráficos en 2D. Además, contiene un paquete que presenta un componente en vivo, implementando una aplicación de zoom y desplazamiento y del gráfico. Está escrito para el JDK 1.1 o superior y puede ser utilizado en MS IE applets.

**BIRT:** es un código abierto basado en Eclipse que se integra con sus aplicaciones Java / aplicaciones Java EE para producir informes de calidad. Además permite la creación de gráficos. Eclipse.org.

**Otras:** E-Gantt, Cewolf, iReport, JOpenChart.

♦ **Más comerciales:**

**Google Chart Tools:** El API de Google Chart permite generar de forma dinámica gráficos con una cadena de dirección URL. Estos gráficos pueden ser insertados en una página web, o descargar la imagen para uso local o en línea, aunque esto tal vez no sea siempre lo más deseado.

## 6.- CONCLUSIONES

Una vez finalizado el presente manual, motivo por el cual se ha llevado a cabo la documentación de la presente memoria, me gustaría reflejar cuáles han sido los objetivos iniciales cubiertos y aquellos aspectos mejorables en futuras versiones.

Procedemos pues, a reflejar los objetivos marcados inicialmente, indicando en cada caso si ha sido alcanzado o el motivo que lo ha impedido.

1. Posibilidad de elección del problema entre varios ofertados (*No Alcanzado, modificado por que sea el usuario quien introduzca los datos a representar*)

2. Descripción (*Alcanzado, se ofrece una amplia descripción de todos y cada uno de los gráficos que se pueden implementar, así como la manera de hacerlo*)

3. Estudio de los datos a tratar (*Alcanzado, se lleva a cabo el correspondiente control de errores para evitar situaciones imprevistas*)

4. Representación gráfica (*Alcanzado, la aplicación demostrativa permite la representación gráfica de los tipos más básicos que se pueden implementar haciendo uso de la librería*)

5. Comparativa (*Alcanzado, se incluye una sección de Alternativas de librerías tanto de código abierto como comerciales. La aplicación demostrativa permite la selección de varios tipos de gráficos que permiten la comparativa entre ellos en la representación de datos*)

6. Interpretación de los resultados obtenidos (*No alcanzado, se recomienda que el usuario de la aplicación demostrativa cuente con los conocimientos oportunos que le permitan la interpretación de los datos mostrados en los gráficos*)

Aunque han quedado objetivos por terminar de afinar, debido básicamente a la falta de tiempo, si podemos decir en términos generales que se haya logrado el alcance que se propuso y que fue el siguiente:

1. Requisitos previos. (*Alcanzado*)

2. Dónde se encuentra la librería y cómo descargarla. (*Alcanzado*)

3. Manejo básico. (*Alcanzado*)

4. Ejemplos de algunas gráficas que implementa. (*Alcanzado*)

5. Comparativa con otras librerías gráficas. (*Alcanzado*)

6. Posibles alternativas al uso de la librería o sus funciones (competencia) (*Alcanzado*)

7. Posibles problemas en su uso o limitaciones. (*Alcanzado*)

8. Aplicaciones destacadas. (*Alcanzado*)

Como mejoras para futuras versiones cabe mencionar la implementación de una clase que permita guardar el gráfico generado en formato pdf y de opción al usuario de seleccionar la ubicación de dicho archivo.

## 7.- BIBLIOGRAFÍA

### Documentos:


- David Gilbert, “The JFreeChart Developer Guide”, Versión 1.0.12, 2 de Enero de 2009.


### Webs:


- Web oficial de JFreeChart:

http://www.jfree.org/jfreechart/

- Java doc de JFreeChart:

http://www.jfree.org/jfreechart/api/javadoc/index.html

- Como instalar las librerias descargadas:

http://www.redribera.es/formacion/tutoriales/viewfile.html?file=javaSeries0108-3-2.xml,

- Definiciones teóricas:

http://es.wikipedia.org
http://www.opensource.org/osd.html

- Alternativas código abierto:

http://jcckit.sourceforge.net/
http://quies.net/java/math/plot/
http://approximatrix.com/products/openchart2/
http://ptolemy.eecs.berkeley.edu/java/ptplot5.1p1/ptolemy/plot/doc/index.htm
http://www.jrobin.org/index.php/Main_Page
http://jcharts.sourceforge.net/
http://jchart2d.sourceforge.net/index.shtml
http://thundergraph.sourceforge.net/

- Alternativas comerciales:

http://mundojava.blogspot.com.es/2010/04/alternativas-para-hacer-analisis.html

## ANEXO I: FIGURAS

A continuación, se incluye una sucesión, por orden de aparición, de todas y cada una de las figuras mencionadas en la memoria.

**Figura 1 .-** *Formulario de propiedades al instalar las librerías*

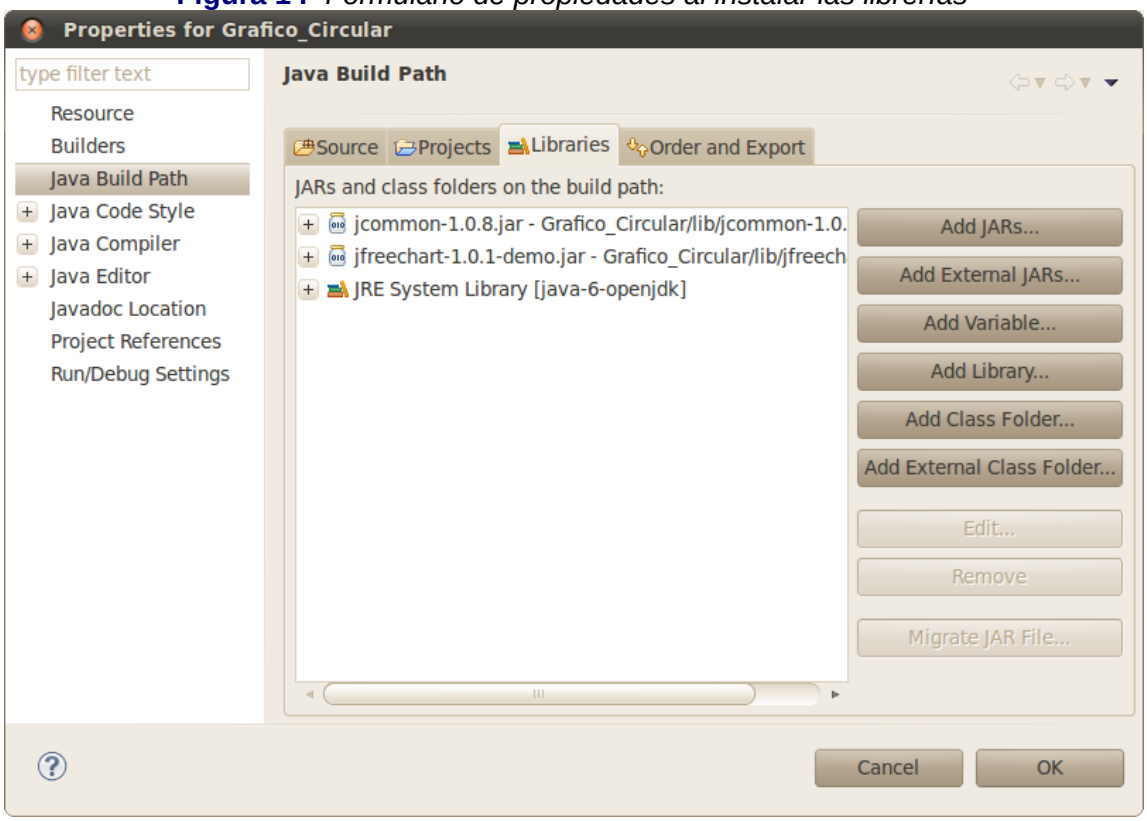



**Figura 2 .-** *Demostración de Gráfico Circular*

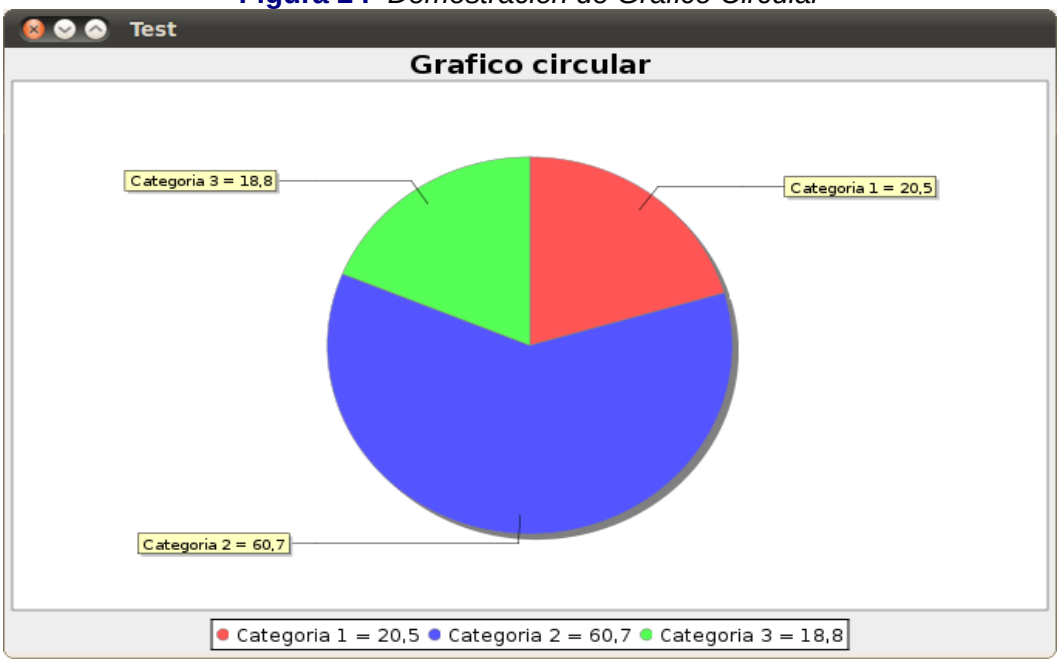


**Figura 3.-** *Demostración de Gráfico de Barras*

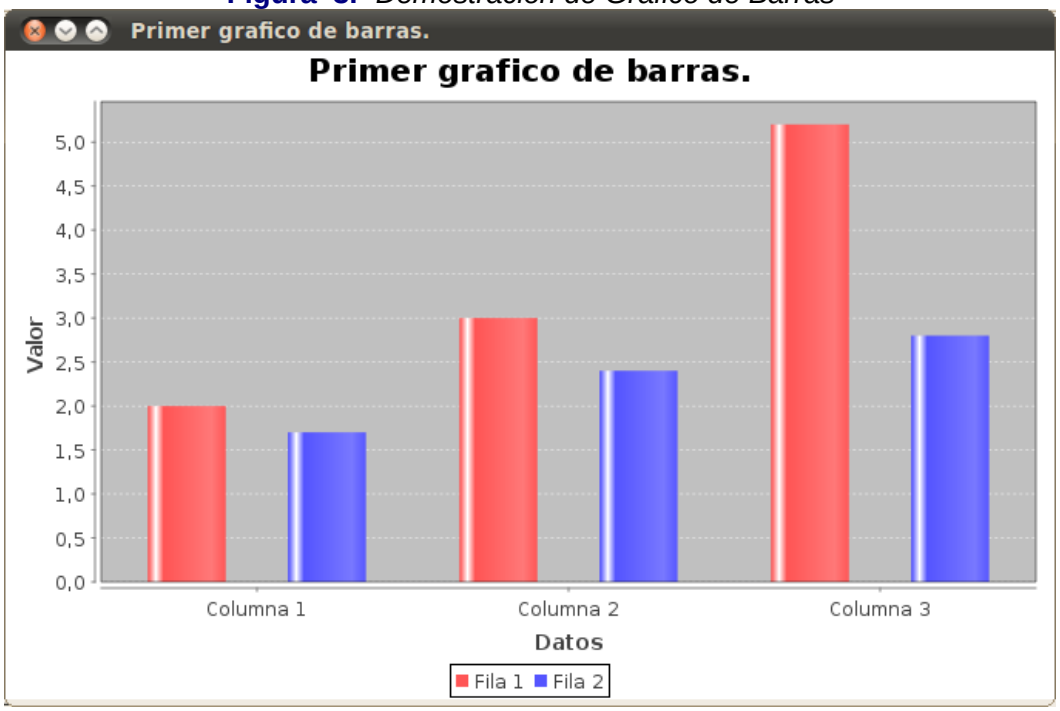



**Figura 4.-** *Demostración de Gráfico de Líneas*

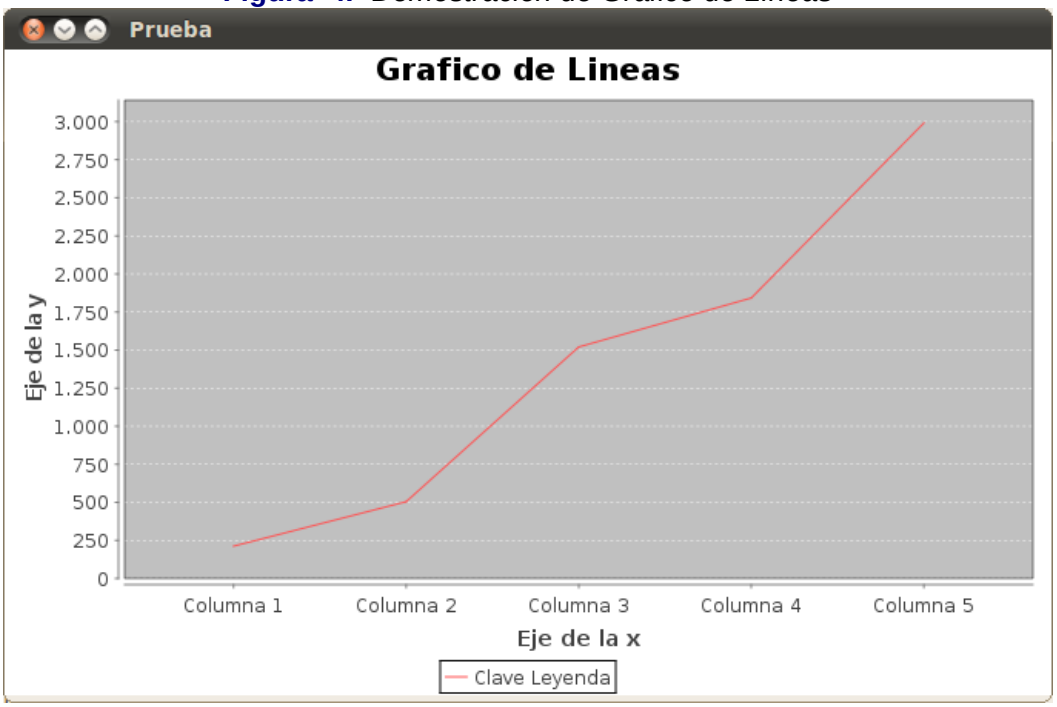


**Figura 5.-** *Asignar colores a una sección en un Gráfico Circular*

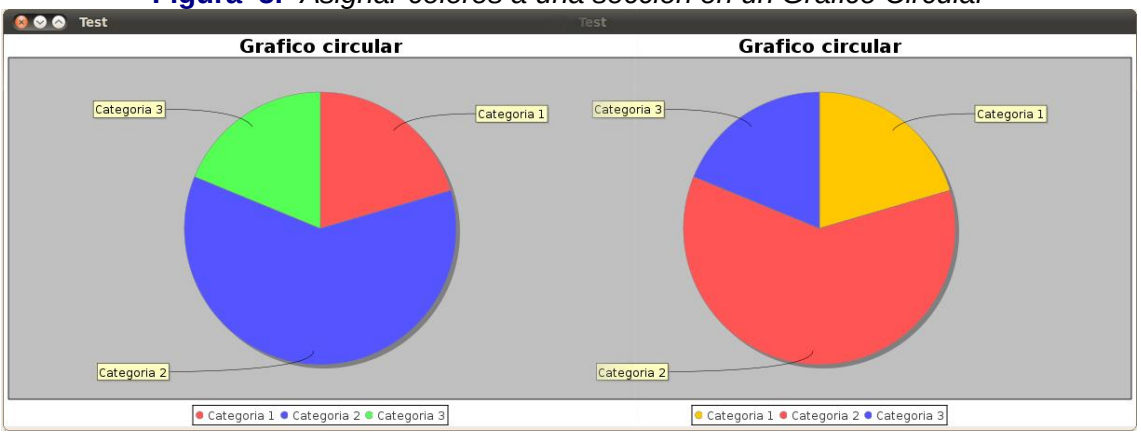



**Figura 6.-** *Color a línea exterior en un Gráfico Circular*


**Figura 7.-** *Gráfico Circular 3D*

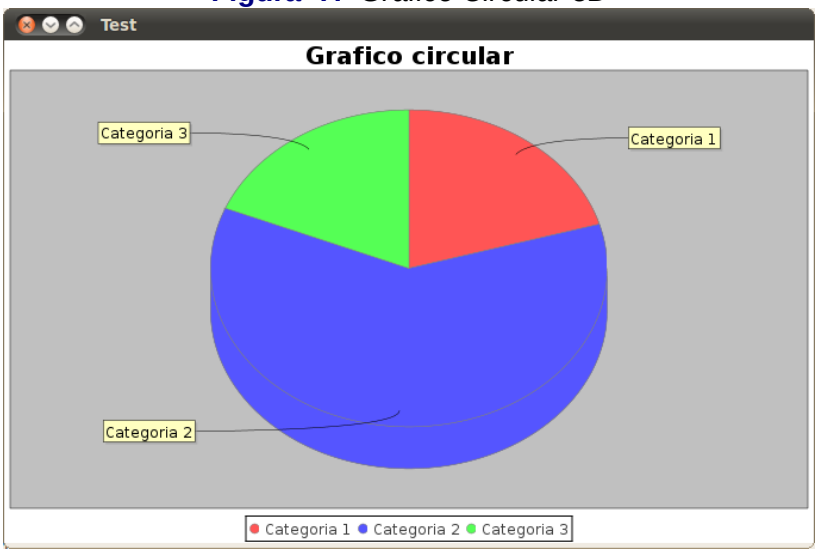



**Figura 8.-** Transparencia en *Gráfico Circular 3D*

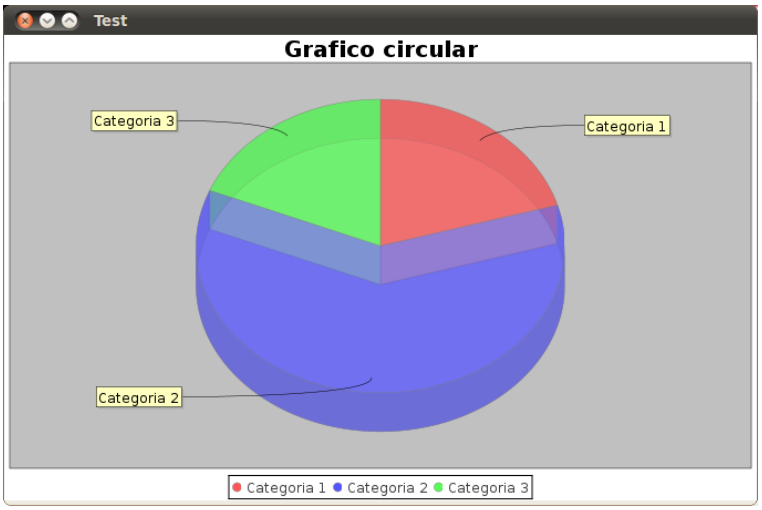



**Figura 9.-** Imagen de prueba a aplicar como fondo

**Figura 10.-** Imagen de prueba aplicada como fondo

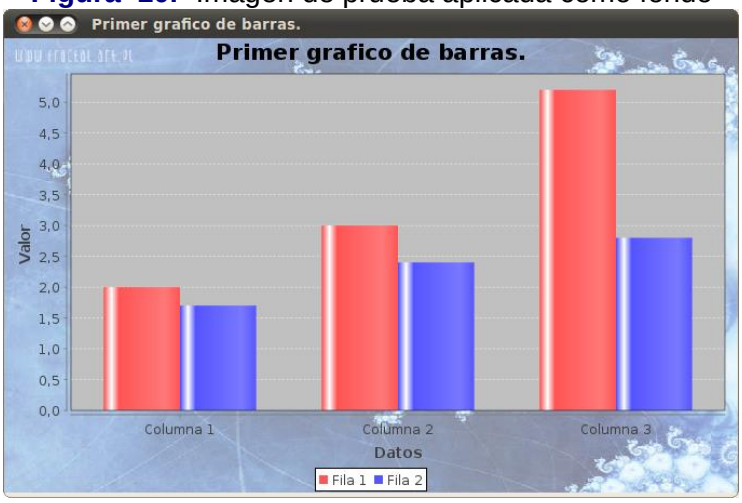



**Figura 11.-** Imagen de fondo optimizada

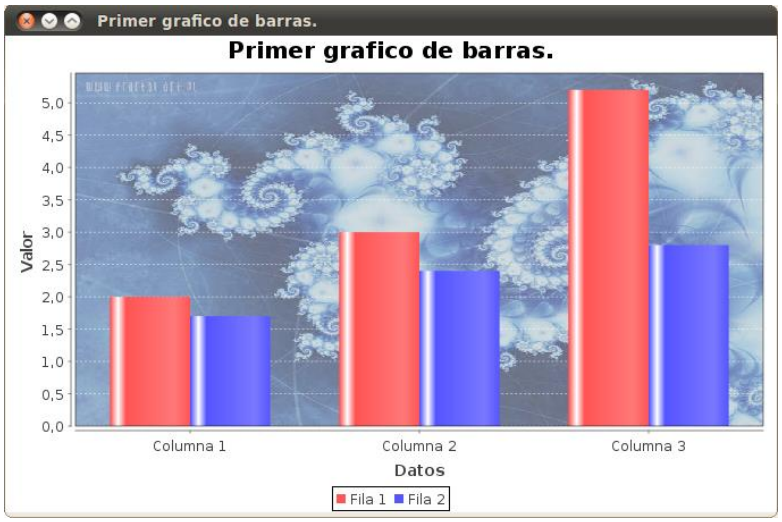



**Figura 12.-** Gráfico de Barras personalizado

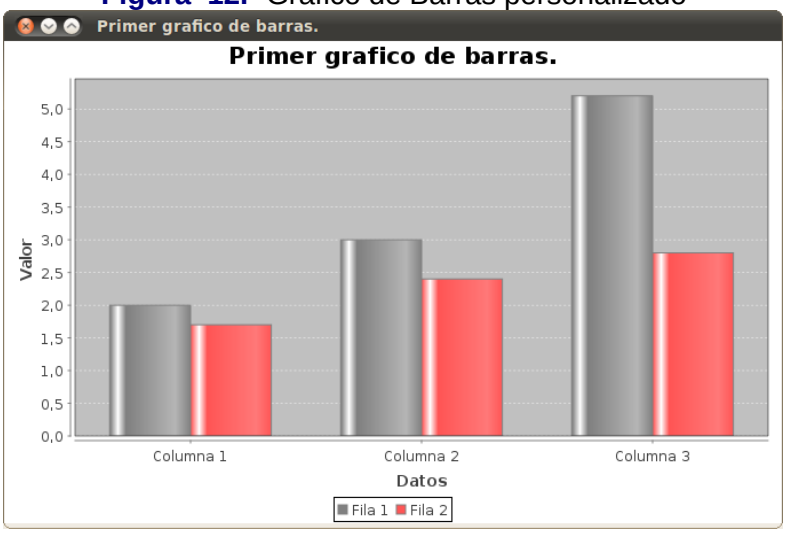
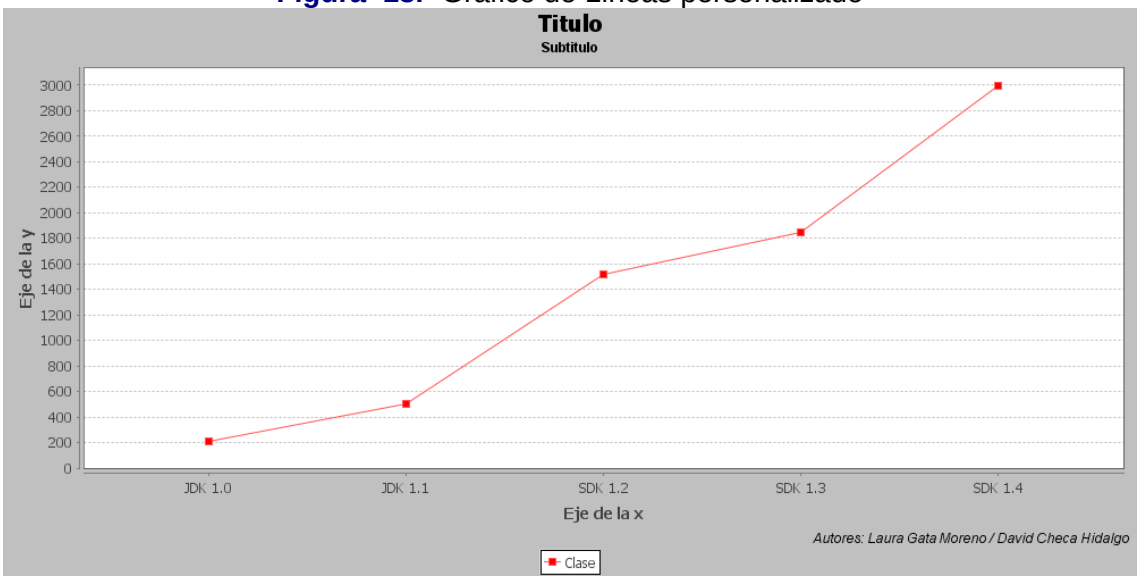



**Figura 13.-** Gráfico de Líneas personalizado



## ANEXO II: TABLAS

De aquí en adelante se incluye una sucesión, por orden de aparición, de todas y cada una de las tablas mencionadas en la memoria.

**Tabla 1 .-** *Datos para el ejemplo del gráfico de barras*

| | Columna 1 | Columna 2 | Columna 3 |
|---|---|---|---|
| **Fila 1** | 2.0 | 3.0 | 5.2 |
| **Fila 2** | 1.7 | 2.4 | 2.8 |

**Tabla 2.-** *Características de JCCKIT*

| | Características | Soporta | Dispositivos | Ofrece | Versiones |
|---|---|---|---|---|---|
| **JCCKit** | Apenas requiere espacio libre en disco (< 100 Kb) | Logaritmos | Gráficos AWT | Clases compiladas para su uso inmediato | Inicial: JCCKit V0.9 lanzada el 12 de abril de 2003 |
| | Altamente configurable | Etiquetas no numéricas | Gráficos 2D AWT (Requiere Java 2) | Código fuente | Final: JCCKit V1.1 lanzada el 18 de diciembre de 2004 |
| | Extensible | Diferentes estilos de líneas, grosores y colores | Creación de imágenes fuera de pantalla (Requiere J2SE 1.4 o superior) | Ejemplos | |
| | Actualizable automáticamente | Diferentes símbolos | SVG (Gráficos vectoriales) | Guía del usuario | |
| | Permite una programación fácil de gráficos | Diferentes tipos de letra, fuentes, colores y orientaciones del texto | | Documentación de la API | |
| | Realiza cambio de escala automática en caso de producirse cambios de tamaño | Barras de error horizontales y verticales | | | |
| | Permite la presentación de los datos en una página web sin necesidad de tener conocimientos del lenguaje de programación Java | Gráficos horizontales, verticales y de barras apiladas | | | |
| | Genera una leyenda automática | | | | |

**Tabla 3.-** *Características de QN Plot*

| | Características | Ofrece | Versiones |
|---|---|---|---|
| QN Plot | Software Libre (BSD "Berkeley Software Distribution") | Documentación de la API | Inicial: QN Plot V1.1 |
| | Alto rendimiento | Código fuente | Final: QN Plot V1.6 |
| | Todas las clases son completamente seguras para subprocesos | | |

**Tabla 4.-** *Características de JOpenChart*

| | Características | Soporta | Versiones |
|---|---|---|---|
| JOpenChart | Permite dibujar diferentes tipos de gráficos como de líneas, barras y circulares | Logaritmos | Final: JOpenChart V0.94 es del 4 de noviembre de 2002 |
| | Incluye las clases necesarias para encapsular todas las partes típicas de un gráfico (leyendas, títulos, ejes…) y conjuntos de datos | Diferentes clases de modelos de datos numéricos | |
| | Buen diseño | Valores no numéricos para los ejes | |
| | Estructuras de clases comprensibles | Datos dinámicos | |
| | Interfaces elegantes | Resultados de una consulta SQL | |
| | Emplea clases abstractas | Polinomios de interpolación spline | |
| | Fácilmente extensible | Diagramas de exportadores de archivos de imagen (png, jpeg) | |
| | Muestra cómo integrar gráficos en applets, aplicaciones Swing con Servlets o JComponent | | |

**Tabla 5.-** *Características de Openchart2*

| | Características | Ofrece | Versiones |
|---|---|---|---|
| **Openchart2** | Es de distribución libre | Documentación de la API | Más reciente: Approximatrix 1.4.3 |
| | Está licenciada bajo GNU Lesser GPL | Código fuente | |
| | Dispone de documentación para desarrolladores | | |
| | Es un proyecto de código abierto | | |

**Tabla 6.-** *Características de PtPlot*

| | Características | Soporta | Versiones |
|---|---|---|---|
| **PtPlot** | Integrable en applets o aplicaciones | Logaritmos | Inicial: PtPlot V1.0, lanzada el 31 de octubre de 1997 |
| | Etiquetado automático o manual de los ejes | Gráficos animados | Más reciente: PtPlot V5.8, lanzada el 5 de octubre de 2012 |
| | Marcas de graduación automáticas o manuales | Zoom infinito | |
| | | Estilos de gráficos: de líneas, de dispersión, de barras, etc | |
| | | Estilos de punto: ninguno, puntos y marcas únicas | |
| | | Conjuntos de datos múltiples | |
| | | Leyenda | |
| | | Color o trazado en blanco y negro | |
| | | Barras de error | |
| | | Gráficos editables | |
| | | PlotML y lenguaje XML para la edición de gráficos | |
| | | Compatibilidad con pxgraph (Programa antiguo) | |

## ANEXO III: Código ejemplo (GraficoCircular.java)

```
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.data.general.DefaultPieDataset;

public class GraficoCircular{

	public static void main(String[] args) {

	// Creamos el dataset
	DefaultPieDataset dataset = new DefaultPieDataset();
	dataset.setValue("Categoria 1", 20.5);
	dataset.setValue("Categoria 2", 60.7);
	dataset.setValue("Categoria 3", 18.8);

	// Creamos el gráfico
	JFreeChart chart = ChartFactory.createPieChart(
		"Grafico circular",
		dataset,
		true,
		true,
		false  );

	// Mostramos el gráfico
	ChartFrame frame = new ChartFrame("Test", chart);
		frame.pack();
		frame.setVisible(true);
		}
	}
```

## ANEXO IV: Código ejemplo (EjemploBarras.java)

```
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;

public class EjemploBarras {

	public static void main(String[] args) {

	// Creamos el dataset
	DefaultCategoryDataset dataset = new DefaultCategoryDataset();

	dataset.addValue(2.0, "Fila 1", "Columna 1");
	dataset.addValue(3.0, "Fila 1", "Columna 2");
	dataset.addValue(5.2, "Fila 1", "Columna 3");
	dataset.addValue(1.7, "Fila 2", "Columna 1");
	dataset.addValue(2.4, "Fila 2", "Columna 2");
	dataset.addValue(2.8, "Fila 2", "Columna 3");

	// Creamos el gráfico
	JFreeChart chart = ChartFactory.createBarChart("Primer grafico de barras." ,
	"Datos","Valor",dataset, PlotOrientation.VERTICAL, true, true, false);

	// Mostramos el gráfico
	ChartFrame frame = new ChartFrame("Primer grafico de barras.",chart);

	frame.pack();
	frame.setVisible(true);
	}
}
```

## ANEXO V: Código ejemplo (GraficoLineas.java)

```
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.chart.plot.PlotOrientation;


public class GraficoLineas {

	public static void main(String[] args) {

	// Creamos el dataset
	DefaultCategoryDataset dataset = new DefaultCategoryDataset();

	dataset.addValue(212, "Clave Leyenda", "Columna 1");
	dataset.addValue(504, "Clave Leyenda", "Columna 2");
	dataset.addValue(1520, "Clave Leyenda", "Columna 3");
	dataset.addValue(1842, "Clave Leyenda", "Columna 4");
	dataset.addValue(2991, "Clave Leyenda", "Columna 5");

	//Creamos el gráfico
	JFreeChart chart = ChartFactory.createLineChart(
	"Grafico de Lineas",
	"Eje de la x",
	"Eje de la y",
	dataset,
	PlotOrientation.VERTICAL,
	true,
	true,
	false);

	// Mostramos el gráfico
	ChartFrame frame = new ChartFrame("Prueba", chart);

	frame.pack();
	frame.setVisible(true);
	}
}
```

## ANEXO VI: Código de la interfaz ejemplo (IGrafico.java)

```
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;


/**
 * Interfaz implementada para las clases de gráficos creadas para este trabajo
 * y que define así una convención de nombrado.
 *
 * @author David Checa Hidalgo
 *
 */
public interface IGrafico {

	/**
	 * Crea el objeto JFreeChart encargado de representar el gráfico por pantalla
	 *
	 * @return el objeto correctamente inicializado
	 *
	 */
	public JFreeChart createChart();

	/**
	* Crea el frame o ventana donde se imprime el gráfico
	*
	* @param jfcChart el objeto de tipo JFreeChart con los datos del gráfico
	a representar.
	 * @return el frame o ventana correctamente creada
	 */
	public ChartFrame createFrame(JFreeChart jfcChart);

	/**
	 * Define los valores del dataset del gráfico correspondiente.
	 */
	public void defineValues();

	/**
	 * Define los valores del dataset y los nombres de las categorías del
	 * gráfico correspondiente.
	 */
	public void defineValuesAndNames();

	/**
	 * Hace visible la venta con el correspondiente gráfico
	 *
	 * @param cfFrame el frame o ventana a representar
	 */
	public void setFrameVisible(ChartFrame cfFrame);
}
```

## ANEXO VII: Código de la clase ejemplo (Errors.java)

```
import java.util.Scanner;


/**
 * Clase creada para el control de errores en la introducción de
 * parámetros por teclado.
 *
 * @author David Checa Hidalgo
 *
 */
public class Errors {

	/**
	 * Controla que el contenido de strNname sea si o no sin atender
	 * a mayúsculas o minúsculas, solicitando un dato mientras que
	 * no se introduzca si o no.
	 *
	 * @param strName variable cadena para la que comprobaremos lo
	 * descrito
	 * @param strMessage mensaje que se imprime por pantalla en caso
	* de error
	 *
	 * @return la variable strName conteniendo si o no
	 *
	 */

	static public boolean yesNoError(String strName, String strMessage){

	Scanner strRead = new Scanner(System.in);

	if(!strName.toLowerCase().equals("si") &&
	!strName.toLowerCase().equals("no")){

		do{
			System.out.println("Error al introducir el dato.");
			System.out.println(strMessage);
			strName = strRead.nextLine();
			}while(!strName.toLowerCase().equals("si")
&&		 !strName.toLowerCase().equals("no"));
		}

		if(strName.toLowerCase().equals("si"))
			return true;
		else
			return false;
	}
```

```
/**
 * Controla que el dato que se lea por teclado sea un entero,
 * solicitando un dato mientras que no sea convertible a entero.
 *
 * @param strMessage mensaje que se imprime por pantalla en caso
 *   de error
 * @return entero correctamente introducido
 *
 */

static public int readInteger(String strMessage){

	int iCategories=0;
	Scanner scRead = new Scanner(System.in);
	boolean bFlag = true;

	do{
		bFlag=false;
		try{
			System.out.println(strMessage);
			iCategories = scRead.nextInt();
		}
		catch(Exception InputMismatchException){
			System.out.println("Error al introducir el dato. El número
			ha de ser un entero.\n");
			scRead.nextLine();
			bFlag = true;
		}
	}while(bFlag);

	return iCategories;
}
```

```
/**
 * Controla que el dato que se lea por teclado sea un entero
 * positivo, solicitando un dato mientras que no sea convertible
 * a entero y/o no sea positivo.
 *
 * @param strMessage mensaje que se imprime por pantalla en caso
 * de error
 * @return entero correctamente introducido
 *
 */

static public int readPositiveInteger(String strMessage){

	int iCategories=0;
	Scanner scRead = new Scanner(System.in);
	boolean bFlag = true;

	do{
		bFlag=false;
		try{
			System.out.println(strMessage);
			iCategories = scRead.nextInt();
		}
		catch(Exception InputMismatchException){
			System.out.println("Error al introducir el dato. El número
			ha de ser un entero positivo.\n");
			scRead.nextLine();
			bFlag = true;
		}
	}while(bFlag || iCategories<0);

	return iCategories;
}
```

```
/**
 * Controla que el dato que se lea por teclado sea un double,
 * solicitando un dato mientras que no sea convertible a double.
 *
 * @param strMessage mensaje que se imprime por pantalla en caso
 *  de error
 * @return double correctamente introducido
 *
 */

static public double readDouble(String strMessage){

	double dCategories=0;
	Scanner scRead = new Scanner(System.in);
	boolean bFlag = true;

	do{
		bFlag=false;
		try{
			System.out.println(strMessage);
			dCategories = scRead.nextDouble();
		}
		catch(Exception InputMismatchException){
			System.out.println("Error al introducir el dato. El número
			ha de ser un double o un entero convertible.\n");
			scRead.nextLine();
			bFlag = true;
		}
	}while(bFlag);

	return dCategories;
}
```

```
/**
 * Controla que el dato que se lea por teclado sea un double
 * positivo, solicitando un dato mientras que no sea convertible
 * a double.
 *
 * @param strMessage mensaje que se imprime por pantalla en caso
 *  de error
 * @return double correctamente introducido
 *
 */

static public double readPositiveDouble(String strMessage){

	double dCategories=0;
	Scanner scRead = new Scanner(System.in);
	boolean bFlag = true;

	do{
		bFlag=false;
		try{
			System.out.println(strMessage);
			dCategories = scRead.nextDouble();
		}
		catch(Exception InputMismatchException){
			System.out.println("Error al introducir el dato. El número
			ha de ser un double o un entero convertible.\n");
			scRead.nextLine();
			bFlag = true;
		}
	}while(bFlag || dCategories<0);

	return dCategories;
}
```

```
/**
 * Método que comprueba que los valores introducidos como % de
 * las categorías del gráfico circular sumen 100% en su totalidad,
 * si esto no se cumple se piden de nuevo. Al acabar la función, el
 * gráfico se encuentra con su dataset correctamente inicializado.
 *
 * @param dValues valores de las categorías del gráfico circular
 * @param iNumber entero que indica el número de categorías
 *
 */

static public void errorPieValues(Double[] dValues,int iNumber){

        double dSuma=0;
        boolean dFlag=false;

        do{
                dFlag=false;
                for(int i=0; i<iNumber; i++)
                        dSuma+=dValues[i];

                if(dSuma<100){
                        System.out.println("El total de las porciones del gráfico ha
                        de sumar el 100%");
                        dFlag=false;
                        for(int i=0; i<iNumber; i++){
                                dValues[i]=Errors.readDouble("Introduce el valor
                                de la categoría "+ (i+1)+" :");
                        }
                }
                dSuma=0;
        }while(dFlag);

}
```

```
	/**
	* Método que comprueba si se ha introducido correctamente la orientación para
	* el gráfico de barras o de líneas: horizontal o vertical sin atender a mayúsculas
	* o minúsculas, solicitando un dato mientras que no se introduzca horizontal o
	* vertical.
	 *
	 * @param strName variable cadena para la que comprobaremos lo descrito
	 * @param strMessage mensaje que se imprime por pantalla en caso de error
	 * @return varible strName correctamente con horizontal o vertical.
	 */

	static public String Orientation(String strName, String strMessage){

		Scanner scRead = new Scanner(System.in);

		if(!strName.toLowerCase().equals("horizontal")
&&		!strName.toLowerCase().equals("vertical")){
			System.out.println("Error al introducir el dato.");
			do{
				System.out.println(strMessage);
				strName = scRead.nextLine();
			}while(!strName.toLowerCase().equals("horizontal") &&
			!strName.toLowerCase().equals("vertical"));
		}

		if(strName.toLowerCase().equals("horizontal"))
			return "horizontal";
		else
			return "vertical";
	}

}
```

## ANEXO VIII: Código para el gráfico circular (PieGraphic.java)

```
import java.util.Scanner;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.data.general.DefaultPieDataset;

/**
 * Clase para la implementación del tipo de dato PieGraphic el cual
 * define a un gráfico circular e implementa los métodos necesarios
 * para su tratamiento.
 *
 * El tipo de dato PieGraphic creado en esta clase será denominado pg
 * a la hora del nombrado en notación húngara, así como:<br/>
 * DefaultPieDataset será dpd<br/>
 * JFreeChart será jfc<br/>
 * ChartFrame será cf
 *
 * @author David Checa Hidalgo
 *
 */
public class PieGraphic implements IGrafico {

	private DefaultPieDataset dpdDataset;

	/**
	 * Constructor de la clase PieGraphic y cuya función es
	 * reservar memoria para la variable privada dpdDataset.
	 */

	public PieGraphic(){
		this.dpdDataset = new DefaultPieDataset();
	}
```

```
	/**
	 * Crea el objeto de tipo JFreeChart que es el encargado de
	 * dibujar las gráficas y que posteriormente situaremos en
	 * una ventana.
	 *
	 * @return el objeto de tipo JFreeChart inicializado con los datos
	 * y la correspondiente personalización.
	 *
	 */
	public JFreeChart createChart() {
		Scanner scRead = new Scanner(System.in);
		String strName = null;
		boolean bLegend = false;

		System.out.println("¿Desea leyenda? (Si|No)");
		strName = scRead.nextLine();

		bLegend = Errors.yesNoError(strName,"¿Desea leyenda? (Si|No)");

		System.out.println("Introduce el titulo del grÃ¡fico: ");
		strName = scRead.nextLine();

		JFreeChart jfcChart = ChartFactory.createPieChart( strName
,		  this.dpdDataset, bLegend, true, false);
		return jfcChart;
	}


	/**
	 * Este método se encarga de la creación del frame o ventana en el que
	 * mostraremos posteriormente nuestro gráfico circular.
	 *
	 * @param jfcChart recibe el objeto de tipo JFreeChart con los datos
	 * del gráfico a representar.
	 * @return el frame inicializado con los datos del gráfico y el
	 * título de la ventana
	 */

	public ChartFrame createFrame(JFreeChart jfcChart) {
		Scanner scRead = new Scanner(System.in);
		String strName = null;

		System.out.println("Introduce el título de la ventana: ");
		strName = scRead.nextLine();

		ChartFrame cfFrame = new ChartFrame(strName, jfcChart);
		return cfFrame;
	}
```

```
/**
 * Este método se encarga de inicializar el dataset con el
 * conjunto de datos que queremos representar en nuestro
 * gráfico circular con nombres por defecto en las categorías.
 *
 * @return el dataset inicializado con los valores del gráfico
 * a representar
 *
 */
public void defineValues() {
	int iCategories;

	iCategories = Errors.readPositiveInteger("Introduce el número
	 de categorías :");

	if(iCategories<1)
		do{
			System.out.println("El número de categorías no puede
			 ser menor que 1.");
			iCategories = Errors.readPositiveInteger("Introduce
			 el número de categorías :");
		}while(iCategories<1);

	Double[] dValues = new Double[iCategories];

	for(int i=0; i<iCategories; i++){
		dValues[i]=Errors.readPositiveDouble("Introduce el valor de
		la categoría "+ (i+1)+" :");
	}

	Errors.errorPieValues(dValues,iCategories);

	for(int i=0; i<iCategories; i++){
		this.dpdDataset.setValue("Categoría "+ (i+1),dValues[i]);
	}
}
```

```
	/**
	 * Método encargado de activar la impresión por pantalla
	 * de nuestro gráfico.
	 *
	 * @param cfFrame el frame o ventana correctamente inicializada
	 *
	 */
	public void setFrameVisible(ChartFrame cfFrame) {
		cfFrame.pack();
		cfFrame.setVisible(true);
	}

	/**
	 * Este metodo se encarga de inicializar el dataset con el
	 * conjunto de datos que queremos representar en nuestro
	 * gráfico circular dándole un nombrado personalizado a
	 * las categorías.
	 *
	 * @return el dataset inicializado con los valores del gráfico
	 * a representar
	 */
	public void defineValuesAndNames() {
		int iCategories;
		Scanner scRead = new Scanner(System.in);

		iCategories = Errors.readPositiveInteger("Introduce el número
		 de categorías :");

		if(iCategories<1)
			do{
				System.out.println("El número de categorías no puede
				 ser menor que 1.");
				iCategories = Errors.readPositiveInteger("Introduce
				 el número de categorías :");
			}while(iCategories<1);

		Double[] dValues = new Double[iCategories];

		String[] strNames = new String[iCategories];

		for(int i=0; i<iCategories; i++){
			scRead.nextLine();
			System.out.println("Introduzca el nombre de la categoría
			 "+ (i+1)+" :");
			strNames[i] = scRead.nextLine();
			dValues[i]=Errors.readPositiveDouble("Introduzca el valor de la
			 categoría "+ (i+1)+" :");
		}

		for(int i=0; i<iCategories; i++){
			this.dpdDataset.setValue(strNames[i],dValues[i]);
		}
	}
}
```

## ANEXO IX: Código para el gráfico de barras (BarsGraphic.java)

```
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;
import java.util.Scanner;

/**
 * Clase para la implementación del tipo de dato BarsGraphic el cual
 * define a un gráfico de barras e implementa los métodos necesarios
 * para su tratamiento.
 *
 * El tipo de dato BarsGraphic creado en esta clase será denominado bg
 * a la hora del nombrado en notación húngara, así como:<br/>
 * DefaultCategoryDataset será dcd<br/>
 * JFreeChart será jfc<br/>
 * ChartFrame será cf
 *
 * @author David Checa Hidalgo
 *
 */
public class BarsGraphic implements IGrafico{

	private DefaultCategoryDataset dcdDataset;

	/**
	 * Constructor de la clase BarsGraphic y cuya función es
	 * reservar memoria para la variable privada dcdDataset.
	 */

	public BarsGraphic(){
		this.dcdDataset = new DefaultCategoryDataset();
	}
```

```
	/**
	 * Crea el objeto de tipo JFreeChart que es el encargado de
	 * dibujar las gráficas y que posteriormente situaremos en
	 * una ventana.
	 *
	 * @return el objeto de tipo JFreeChart inicializado con los datos
	 * y la correspondiente personalización.
	 *
	 */
	public JFreeChart createChart() {
		Scanner scRead = new Scanner(System.in);
		String strName = null,axeX,axeY, strOrientation;
		boolean bLegend = false;
		JFreeChart jfcChart;

		System.out.println("¿Desea leyenda? (Si|No)");
		strName = scRead.nextLine();

		bLegend = Errors.yesNoError(strName,"¿Desea leyenda? (Si|No)");

		System.out.println("Introduce el titulo del gráfico: ");
		strName = scRead.nextLine();
		System.out.println("Introduce el nombre del eje X: ");
		axeX = scRead.nextLine();
		System.out.println("Introduce el nombre del eje Y: ");
		axeY = scRead.nextLine();
		System.out.println("¿Desea orientación vertical u horizontal para
		las columnas? ");
		strOrientation = scRead.nextLine();

		strOrientation =
Errors.Orientation(strOrientation,"¿Desea orientación vertical u
horizontal para las columnas? ");

		if(strOrientation.toLowerCase().equals("vertical"))
			jfcChart = ChartFactory.createBarChart(strName, axeX, axeY ,
			this.dcdDataset, PlotOrientation.VERTICAL, bLegend, false,
			false);
		else
			jfcChart = ChartFactory.createBarChart(strName, axeX, axeY ,
			this.dcdDataset, PlotOrientation.HORIZONTAL, bLegend,false,
			false);

		return jfcChart;
	}
```

```
/**
 * Este método se encarga de la creación del frame o ventana en el que
 * mostraremos posteriormente nuestro gráfico de barras
 * @param jfcChart recibe el objeto de tipo JFreeChart con los datos
 * del gráfico a representar.
 * @return el frame inicializado con los datos del gráfico y el
 * título de la ventana
 */
public ChartFrame createFrame(JFreeChart jfcChart) {
	Scanner scRead = new Scanner(System.in);
	String strName = null;

	System.out.println("Introduce el título de la ventana: ");
	strName = scRead.nextLine();

	ChartFrame cfFrame = new ChartFrame(strName, jfcChart);
	return cfFrame;
}
```

```
/**
 * Este método se encarga de inicializar el dataset con el
 * conjunto de datos que queremos representar en nuestro
 * gráfico de barras con nombres por defecto en las categorías.
 *
 * @return el dataset inicializado con los valores del gráfico
 * a representar
 *
 */
public void defineValues() {
        int iColumns, iRows;

        iRows = Errors.readPositiveInteger("Introduzca el número de
         filas: (número de barras por grupo)");

        if(iRows<1)
                do{
                        System.out.println("El número de filas no puede
                         ser menor que 1.");
                        iRows = Errors.readPositiveInteger("\nIntroduzca
                        el número de filas: (número de barras por grupo)");
                }while(iRows<1);

        iColumns = Errors.readPositiveInteger("Introduzca el número
        de columnas: (número de grupos de barras)");

        if(iColumns<1)
                do{
                        System.out.println("El número de columnas no puede ser
                        menor que 1.");
                        iColumns = Errors.readPositiveInteger("\nIntroduzca el
                        número de columnas: (número de grupos de barras)");
                }while(iColumns<1);

        Double[][] dValues = new Double[iRows][iColumns];

        for(int i=0; i<iColumns; i++){
                for(int j=0; j<iRows; j++){
                        System.out.println("Introduzca los valores del grupo "+
                        (i+1)+" de columnas:");
                        dValues[j][i] = Errors.readDouble("Introduzca el valor de l
                        a columna "+ (j+1) + " :");
                }
        }

        for(int i=0; i<iRows; i++){
                for(int j=0; j<iColumns; j++){
                        this.dcdDataset.addValue(dValues[i][j],"Fila "+
                        (i+1),"Columna "+(j+1));
                }
        }

}
```

```
/**
 * Método encargado de activar la impresión por pantalla
 * de nuestro gráfico.
 *
 * @param cfFrame el frame o ventana correctamente inicializado
 *
 */
public void setFrameVisible(ChartFrame cfFrame) {
        cfFrame.pack();
        cfFrame.setVisible(true);
}
```

```
/**
 * Este método se encarga de inicializar el dataset con el
 * conjunto de datos que queremos representar en nuestro
 * gráfico circular dándole un nombrado personalizado a
 * las categorías.
 *
 * @return el dataset inicializado con los valores del gráfico
 * a representar
 *
 */

public void defineValuesAndNames() {
        int iColumns, iRows;
        Scanner scRead = new Scanner(System.in);

        iRows = Errors.readPositiveInteger("Introduzca el númerode
         filas: (número de barras por grupo)");

        if(iRows<1)
                do{
                        System.out.println("El número de filas no puede ser menor
                        que 1.");
                        iRows = Errors.readPositiveInteger("\nIntroduzca el
                        número de filas: (número de barras por grupo)");
                }while(iRows<1);

        iColumns = Errors.readPositiveInteger("Introduzca el número
         de columnas: (número de grupos de barras)");

        if(iColumns<1)
                do{
                        System.out.println("El número de columnas no puede ser
                        menor que 1.");
                        iColumns = Errors.readPositiveInteger("\nIntroduzca el nú
                        mero de columnas: (número de grupos de barras)");
                }while(iColumns<1);

        Double[][] dValues = new Double[iRows][iColumns];
        String[] strNames = new String[iRows];

        for(int i=0; i<iRows; i++){
                System.out.println("Introduzca el nombre del grupo "+ (i+1)+" de
                columnas:");
                strNames[i] = scRead.nextLine();

                for(int j=0; j<iColumns; j++){
                        System.out.println("Introduzca los valores del grupo "+
                        (i+1)+" de columnas:");
                        dValues[i][j] = Errors.readDouble("Introduzca el valor de l
                        a columna "+ (j+1) + " :");
                }
        }
```

```
                for(int i=0; i<iRows; i++){
                        for(int j=0; j<iColumns; j++){
                                this.dcdDataset.addValue(dValues[i][j],"Fila "+
                                (i+1),strNames[i]);
                        }
                }

        }

}
```

## ANEXO X: Código para el gráfico de líneas (LineGraphic.java)

```
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;
import java.util.Scanner;

/**
 * Clase para la implementación del tipo de dato LineGraphic el cual
 * define a un gráfico de líneas e implementa los métodos necesarios
 * para su tratamiento.
 *
 * El tipo de dato LineGraphic creado en esta clase será denominado lg
 * a la hora del nombrado en notación húngara, así como:<br/>
 * DefaultCategoryDataset será dcd<br/>
 * JFreeChart será jfc<br/>
 * ChartFrame será cf
 *
 * @author David Checa Hidalgo
 *
 */
public class LineGraphic implements IGrafico{

	private DefaultCategoryDataset dcdDataset;

	/**
	 * Constructor de la clase LineGraphic y cuya función es
	 * reservar memoria para la variable privada dcdDataset.
	 */
	public LineGraphic(){
		this.dcdDataset = new DefaultCategoryDataset();
	}
```

```
/**
 * Crea el objeto de tipo JFreeChart que es el encargado de
 * dibujar las gráficas y que posteriormente situaremos en
 * una ventana.
 *
 * @return el objeto de tipo JFreeChart inicializado con los datos
 * y la correspondiente personalización.
 *
 */

public JFreeChart createChart() {
        Scanner scRead = new Scanner(System.in);
        String strName = null,axeX,axeY, strOrientation;
        boolean bLegend = false;
        JFreeChart jfcChart;

        System.out.println("¿Desea leyenda? (Si|No)");
        strName = scRead.nextLine();

        bLegend = Errors.yesNoError(strName,"¿Desea leyenda? (Si|No)");

        System.out.println("Introduce el título del gráfico: ");
        strName = scRead.nextLine();
        System.out.println("Introduce el nombre del eje X: ");
        axeX = scRead.nextLine();
        System.out.println("Introduce el nombre del eje Y: ");
        axeY = scRead.nextLine();
        System.out.println("¿Desea orientación vertical u horizontal para
         las columnas? ");
        strOrientation = scRead.nextLine();

        strOrientation = Errors.Orientation(strOrientation,"¿Desea
         orientación vertical u horizontal para las columnas? ");

        if(strOrientation.toLowerCase().equals("vertical"))
                jfcChart = ChartFactory.createLineChart(strName, axeX, axeY ,
                this.dcdDataset, PlotOrientation.VERTICAL, bLegend, false,
                false);
        else
                jfcChart = ChartFactory.createLineChart(strName, axeX, axeY ,
                this.dcdDataset, PlotOrientation.HORIZONTAL, bLegend,false,
                false);

        return jfcChart;
}
```

```
/**
 * Este método se encarga de la creación del frame o ventana en el que
 * mostraremos posteriormente nuestro gráfico de líneas.
 *
 * @param jfcChart recibe el objeto de tipo JFreeChart con los datos del
 * gráfico a representar.
 * @return el frame inicializado con los datos del gráfico y el
 * título de la ventana
 */
public ChartFrame createFrame(JFreeChart jfcChart) {
	Scanner scRead = new Scanner(System.in);
	String strName = null;

	System.out.println("Introduce el título de la ventana: ");
	strName = scRead.nextLine();

	ChartFrame cfFrame = new ChartFrame(strName, jfcChart);
	return cfFrame;
}
```

```
/**
 * Este método se encarga de inicializar el dataset con el
 * conjunto de datos que queremos representar en nuestro
 * gráfico de líneas con nombres por defecto en las categorías.
 *
 * @return el dataset inicializado con los valores del gráfico
 * a representar
 *
 */
public void defineValues() {
	int iPoints;

	iPoints = Errors.readInteger("Introduzca el número de puntos
	 a representar: ");

	if(iPoints<1)
		do{
			System.out.println("El número de puntos no puede ser
			menor que 1.");
			iPoints = Errors.readPositiveInteger("Introduzca el número
			de puntos a representar: ");
		}while(iPoints<1);

	Double[]dValues = new Double[iPoints];

	for(int i=0; i<iPoints; i++){
			dValues[i] = Errors.readDouble("Introduzca el valor
			"+(i+1)+": ");
	}

	for(int i=0; i<iPoints; i++){

this.dcdDataset.addValue(dValues[i],"Clase","Punto "+(i+1));
	}

}
```

```
/**
 * Método encargado de activar la impresión por pantalla
 * de nuestro gráfico.
 *
 * @param cfFrame el frame o ventana correctamente inicializado
 *
 */
public void setFrameVisible(ChartFrame cfFrame) {
        cfFrame.pack();
        cfFrame.setVisible(true);
}
```

```
	/**
	 * Este método se encarga de inicializar el dataset con el
	 * conjunto de datos que queremos representar en nuestro
	 * gráfico de líneas dándole un nombrado personalizado a
	 * las categorías.
	 *
	 * @return el dataset inicializado con los valores del gráfico
	 * a representar
	 *
	 */

	public void defineValuesAndNames() {
		int iPoints;
		Scanner scRead = new Scanner(System.in);

		iPoints = Errors.readInteger("Introduzca el número de puntos
		 a representar: ");

		if(iPoints<1)
			do{
				System.out.println("El número de puntos no puede ser
				menor que 1.");
				iPoints = Errors.readPositiveInteger("Introduzca el número
				de puntos a representar: ");
			}while(iPoints<1);

		Double[] dValues = new Double[iPoints];
		String[] strNames = new String[iPoints];

		for(int i=0; i<iPoints; i++){
			System.out.println("Introduzca la etiqueta para el valor "+ (i+1)+"
			:");
			strNames[i] = scRead.nextLine();
			dValues[i] = Errors.readDouble("Introduzca el valor "+ (i+1) +
			" :");
		}

		for(int i=0; i<iPoints; i++){
				this.dcdDataset.addValue(dValues[i],"Clase",strNames[i]);
			}
		}
}
```

## ANEXO XI: Código para la aplicación demostrativa (Prueba.java)

```
import java.util.Scanner;

import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;


/**
 * Clase para la prueba de las clases de gráficos creadas para
 * el trabajo de Programación Aplicada.
 * Se implementan unas funciones auxiliares de tipo static y el
 * main del programa.
 *
 * @author David Checa Hidalgo
 *
 */

public class Prueba {

	/**
	 * Realiza el proceso completo de creación de un gráfico circular
	 * así como el control de errores al introducir los parámetros
	 * de entrada solicitados.
	 *
	 * @param pgGrafico un objeto de tipo PieGraphic para trabajar con éste
	 */

	static public void startGraphicPie(PieGraphic pgGrafico){
		String strOpcion = null;
		Scanner scRead = new Scanner(System.in);
		JFreeChart jfcChart;
		ChartFrame cfFrame;
		boolean bCategories;

		System.out.println("¿Desea nombrar las categorías? (Si|No)");
		strOpcion = scRead.nextLine();

		bCategories = Errors.yesNoError(strOpcion,"¿Desea nombrar
		 las categorías? (Si|No)");

		if(bCategories)
			pgGrafico.defineValuesAndNames();
		else
			pgGrafico.defineValues();

		jfcChart = pgGrafico.createChart();
		cfFrame = pgGrafico.createFrame(jfcChart);
		pgGrafico.setFrameVisible(cfFrame);
	}
```

```
/**
 * Realiza el proceso completo de creación de un gráfico de barras
 * así como el control de errores al introducir los parámetros
 * de entrada solicitados.
 *
 * @param pgGrafico un objeto de tipo BarsGraphic para trabajar con éste
 */

static public void startGraphicBars(BarsGraphic bgGrafico){
        String strOpcion = null;
        Scanner scRead = new Scanner(System.in);
        JFreeChart jfcChart;
        ChartFrame cfFrame;
        boolean bCategories;

        System.out.println("¿Desea nombrar las categorías? (Si|No)");
        strOpcion = scRead.nextLine();

        bCategories = Errors.yesNoError(strOpcion,"¿Desea nombrar
         las categorías? (Si|No)");

        if(bCategories)
                bgGrafico.defineValuesAndNames();
        else
                bgGrafico.defineValues();

        jfcChart = bgGrafico.createChart();
        cfFrame = bgGrafico.createFrame(jfcChart);
        bgGrafico.setFrameVisible(cfFrame);
}
```

```
/**
 * Realiza el proceso completo de creación de un gráfico de líneas
 * así como el control de errores al introducir los parámetros
 * de entrada solicitados.
 *
 * @param lgGrafico un objeto de tipo LineChart para trabajar con este
 */

static public void startLineGraphic(LineGraphic lgGrafico){
	String strOpcion = null;
	Scanner scRead = new Scanner(System.in);
	JFreeChart jfcChart;
	ChartFrame cfFrame;
	boolean bCategories;

	System.out.println("¿Desea nombrar las categorías? (Si|No)");
	strOpcion = scRead.nextLine();

	bCategories = Errors.yesNoError(strOpcion,"¿Desea nombrar las
	categorías? (Si|No)");

	if(bCategories)
		lgGrafico.defineValuesAndNames();
	else
		lgGrafico.defineValues();

	jfcChart = lgGrafico.createChart();
	cfFrame = lgGrafico.createFrame(jfcChart);
	lgGrafico.setFrameVisible(cfFrame);
}
```

```
	/**
	 * Main de la clase de prueba que define los gráficos según lo
	 * desee el usuario
	 *
	 * @param strArgs no es necesario ningún argumento
	 *
	 */
	public static void main(String[] strArgs) {
		String strOpcion = null;
		Scanner scRead = new Scanner(System.in);
		boolean bFlag = true;

		while(bFlag){
			System.out.println("Introduce el tipo de gráfico que deseas:\nbarras | circular | lineas");
			System.out.println("Introduce salir para terminar el programa:");
			strOpcion = scRead.nextLine();

			if(strOpcion.toLowerCase().equals("barras")){
				BarsGraphic bgGrafico = new BarsGraphic();
				startGraphicBars(bgGrafico);
			}
			else if(strOpcion.toLowerCase().equals("circular")){
				PieGraphic pgGrafico = new PieGraphic();
				startGraphicPie(pgGrafico);
			}
			else if(strOpcion.toLowerCase().equals("salir")){
				System.out.println("\nAdios.");
				bFlag=false;
			}
			else if(strOpcion.toLowerCase().equals("lineas")){
				LineGraphic lgGrafico = new LineGraphic();
				startLineGraphic(lgGrafico);
			}
			else{
				System.out.println("Error al introducir el tipo de gráfico.\n");
			}
		}

	}

}
```