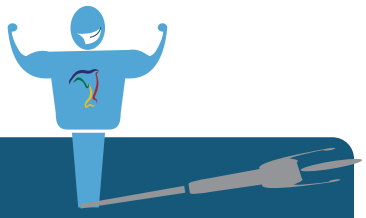
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

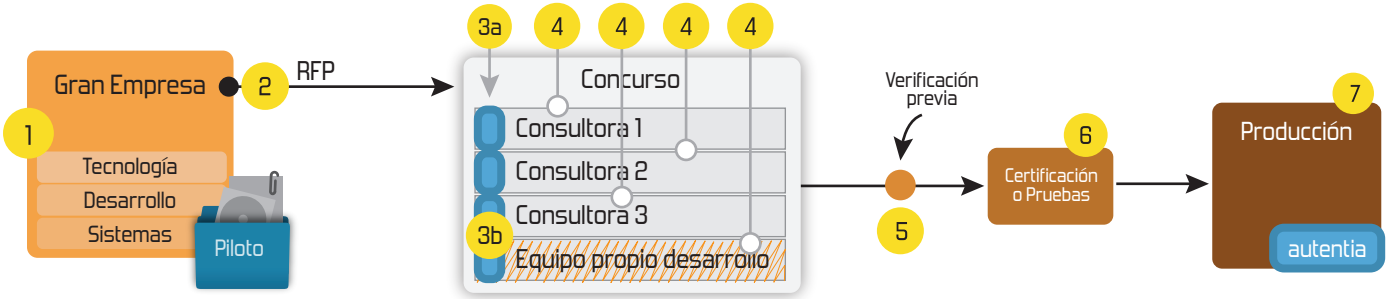
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Lanzado TNTConcept versión 0.8 (10/12/2007)

¿Gestionas tu empresa con hojas de cálculo? ¿No crees que puede haber un modo mejor?

Desde Autentia ponemos a vuestra disposición el software que hemos construido (100% gratuito, con código fuente disponible y sin restricciones funcionales) para nuestra gestión interna, llamado TNTConcept (au**T**e**NT**ia). Construida con las últimas tecnologías de desarrollo Java/J2EE (Spring, JSF, Acegi, Hibernate, Maven, Subversion, etc.) y disponible en licencia GPL, seguro que a muchos profesionales independientes y PYMES os ayudará a organizar mejor vuestra operativa.

Las cosas grandes empiezan siendo algo pequeño ..... Saber más en: http://tntconcept.sourceforge.net/

Tutorial desarrollado por Labarca Gallardo Rhaimanhs Raul: rhaimanhs@yahoo.com.mx

Descargar este documento en formato PDF javaPrinting.pdf

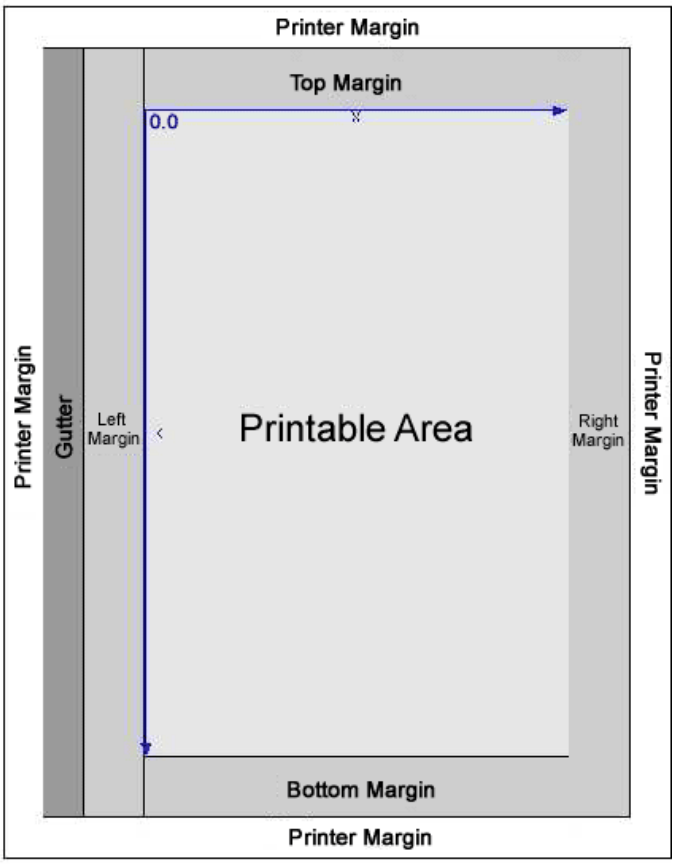
Fecha de creación del tutorial: 2007-12-10

La API Java Printing

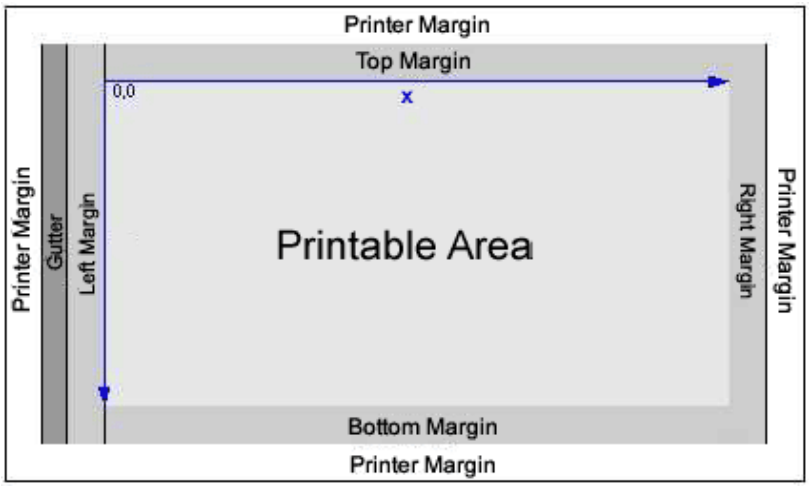
Traducción: Rhaimanhs Labarca
Definición de una página

Antes de que nos involucremos en las tecnicidades del API de impresión, vamos a comenzar definiendo una cierta terminología que se utilizará a lo largo de este documento. Aunque esta terminología puede parecer trivial, ayudará a despejar ciertas confusiones sobre márgenes.

Quizás sabes (de lo contrario lo sabrás ahora) que Gutenberg inventó la primera prensa. En aquella época, él tuvo que crear una terminología para describir la disposición de una página. Aquí está la definición de que dio de una página:



Página en Forma Horizontal



Página en forma Horizontal

En las imágenes anteriores, podemos ver que la página está dividida en varias áreas. Los márgenes de la impresora constituyen la periferia de la página. Ellos son dependientes de la impresora y definen el área mínima a que la impresora necesita alimentar la página. Los márgenes de la impresora no son definibles por el usuario. Raramente usamos o sabemos los tamaños de márgenes de la impresora, pero algunos fabricantes de impresora publican esto en sus manuales. En Java, no se necesita saber estas medidas; el API devuelve las dimensiones del área que puede ser impresa.

Justo al interior de los márgenes de la impresora son los márgenes que definen el contorno de la página. Note que los márgenes izquierdos y derechos extienden la longitud de la página, menos los márgenes superiores e inferiores de impresión. El margen al lado izquierdo (gutter) proporciona un margen adicional que se utilice

sobre todo para encuadernar las páginas en un libro. En una página impresa en ambos lados, el margen puede ser encontrado en el derecho de la página. El API de impresión en sí mismo no soporta este margen. Tan extraño como puede parecer, la API de impresión tampoco puede soportar márgenes. La única manera de fijar márgenes es fijar la localización y el tamaño del área imprimible.

Finalmente, el área en el medio de la página se llama el área imprimible.

Unidades de medida

Al trabajar con la clase Graphics2D, es importante entender la diferencia entre el espacio de dispositivo y el espacio del usuario. En el espacio de dispositivo, se trabaja en píxeles usando la resolución del dispositivo. Un cuadrado de 100 x 100 píxeles dibujados en un dispositivo que tenga una resolución de 1.024 x 768 píxeles no será del mismo tamaño en comparación a la de un dispositivo que tenga una resolución de 1.600 x 1400 píxeles. La razón es simple: porque el segundo dispositivo ofrece más píxeles por pulgada, así el cuadrado aparecerá más pequeño.

El espacio del usuario, por otra parte, permite que pensemos en términos de unidades de medida, sin importar la resolución del dispositivo. Cuando se crea un componente Graphics2D para un dispositivo dado (impresora o pantalla), un valor predeterminado transforma el espacio de usuario al espacio del dispositivo.

En espacio de usuario, el valor predeterminado se fija a 72 coordenadas por pulgada. En vez de pensar en términos de píxeles, se piensa en términos de unidades. Un cuadrado de una pulgada, es en realidad un cuadrado de 72 x 72 unidades. Una página tamaño carta (8,5 por 11 pulgadas) es en realidad un rectángulo de 612 x 792 puntos. Al usar el API de impresión, debemos fijar todas nuestras medidas en unidades ya que todas las clases trabajan en el espacio de usuario.

Sistema de impresión de Java

El sistema de impresión de Java se ha desarrollado considerablemente en sus dos lanzamientos pasados. Comenzando con la versión 1,2, el sistema de impresión permite que se utilice la API Java2D para renderizar una página. Este API permite que cualquier cosa que pueda ser dibujada en la pantalla pueda ser impresa.

Aunque se ha avanzado bastante en el API de impresión, ésta solo soporta la impresora seleccionada actualmente por el usuario en cualquier momento. Java no soporta el Printer Discovery (obtención de una lista de impresoras disponibles y de sus características en una computadora dada). Las impresoras disponibles pueden ser locales o conectadas en red. Al usar el API, no existe ninguna manera para obtener una lista de la impresora a través de codificación; solamente si se exhibe el diálogo de impresión, es posible seleccionar una determinada impresora. Esto es una característica que se echa de menos en el API.

El API de impresión se basa en un modelo del retrollamada, en el que el subsistema de impresión, no su programa, controla cuando se renderiza una página.

Para simplificar un poco esto, digamos que nuestro programa tiene un contrato con el subsistema de impresión para proveer una página dada en un momento dado. El subsistema de impresión puede solicitar a mi programa que renderice una página más de una vez, o renderice las páginas fuera de secuencia. Este modelo proporciona varias ventajas. Primero, enviando las páginas (pensemos que todas nuestra páginas en realidad es una larga pagina, como una tira de papel continuo), en vez de la página entera a la impresora, permite que el programa imprima documentos complejos que requerirían más memoria de la que la impresora dispone. El programa no tiene que saber como imprimir cada tira; necesita solamente saber renderizar una página dada. El API se encarga del resto. En este caso, el subsistema de impresión puede solicitar que una página esté renderizada varias veces dependiendo del número de las tiras requeridas para imprimir totalmente la página. En segundo lugar, si se desea imprimir en orden inversa, entonces mi programa debe proporcionar el medio para imprimir el documento en forma inversa.

Representación de modelos

Hay dos modelos de impresión en Java: Printable jobs y Paginable jobs. Veamos cada uno de ellos.

a) Printable

El modelo Printable job es el más simple de los dos modelos de impresión. Este modelo utiliza solamente un PagePainter (pintor de página) para el documento entero. Las páginas se renderizan en secuencia, comenzando con la página cero. Cuando la última página se imprime, pintor de página (PagePinter) debe devolver el valor NO_SUCH_PAGE. El subsistema de impresión solicitará siempre que el programa renderice las páginas en forma secuencial. Como ejemplo, si al programa se le pide imprimir las páginas cinco a siete, el subsistema de impresión pedirá que todas las páginas sean renderizadas (hasta la séptima página), pero se imprimirá solamente las páginas cinco, seis, y siete. Si la aplicación exhibe una cuadro de diálogo de impresión, el número total de páginas que se imprimirán no será mostrado puesto que es imposible saber por adelantado el número de páginas en el documento usando este modelo.

b) Pageable

El modelo Pageable job ofrece más flexibilidad que el modelo anterior; de hecho, este modelo permite que cada pagina se imprima en la disposición que se desea, con un pintor propio. Por ejemplo si queremos imprimir un documento de 4 páginas con las siguientes disposiciones: Las primeras dos páginas en forma vertical, la 3ra en forma horizontal y la ultima en forma vertical, con este modelo podemos hacerlo. Hacer lo anterior, con el otro modelo, será mas complejo de realizar. Pageable job se utiliza lo más a menudo posible con Book, (libro) que se interpreta como una colección de páginas. Por cierto, una colección de páginas puede tener diversos formatos.

Una Pageable job tiene las características siguientes:

1. Cada página puede tener su propio pintor. Por ejemplo, podemos hacer un pintor para imprimir la página de cubierta, otro pintor para imprimir el contenido, y un tercero para imprimir el documento entero.

2. Se puede fijar un diverso formato de página para cada página en el libro. En un Paginable job, podemos mezclar páginas en orden horizontal como vertical.

3. El subsistema de impresión puede pedir a nuestro programa imprimir las páginas fuera de secuencia, y algunas páginas se pueden saltar en caso de ser necesario. Una vez más no tenemos de que preocuparnos, mientras proporcionemos cualquier página que sea requerida.

4. Paginable job no necesita saber cuántas páginas hay en el documento a imprimir.

Books (Libros)

Esta clase permite que se creen documentos de multiples paginas. Cada página puede tener su propio formato y su propio pintor, dándo la flexibilidad de crear documentos sofisticados. Además la clase Books implementa la interfaz Pageable.

Una Clase Book representa una colección de páginas. Cuando creamos un objeto Book, tal objeto es vacío. Para agregar páginas, utilizamos simplemente uno de los dos métodos append. Los parámetros de este método son un objeto PageFormat, que define las características físicas de la página, y un objeto PagePainter, el cual implementa la interfaz Printable.

Si desconocemos el número de páginas en nuestro documento, simplemente pasamos el valor UNKNOWN_NUMBER_OF_PAGES al método append(). El sistema de impresión encontrará automáticamente el número de páginas llamando a todos los pintores de la página en el libro hasta que recibe el valor NO_SUCH_PAGE.

Definición de la API

Todas las clases requeridas para imprimir son localizadas en el paquete java.awt.print, que es compuesto de tres interfaces y cuatro clases. Las tablas siguientes definen las clases y los interfaces del paquete print.

Nombre	**Tipo**	**Descripción**
Paper	Clase	Esta clase define las características físicas de la página.
PageFormat	Clase	PageFormat define el tamaño de la página y la orientación. Esto también define Paper usar cuando se renderiza una página.
PrinterJob	Clase	Esta clase maneja el trabajo de impresión. Sus responsabilidades incluyen la creación de un trabajo de impresión, desplegar un cuadro de diálogo de impresión cuando sea necesario, y la impresión del documento.
Book	Clase	El libro, representa un documento. Un objeto Book actúa como una colección de páginas. Las páginas incluidas en el Libro pueden tener formatos idénticos o que se diferencian y pueden usar a pintores diferentes.
Pageable	Interfaz	Una implementación de Paginable representa un juego de páginas para ser impresas. El objeto Paginable devuelve el número total de páginas en el juego, así como el PageFormat y el Printable para una página específica. La clase Book implementa esta interfaz.
Printable	Interfaz	Un pintor de página debe implementar la interfaz Printable. Hay sólo un método en este interfaz, print ().
PrinterGraphics	Interfaz	El objeto Graphics implementa esta interfaz. PrinterGraphics proporciona el método getPrinterJob () para obtener el trabajo de impresión (printer job)el trabajo de impresora que está instanciado en el proceso

	de impresión.

- Interfaz Pageable

El interfaz Paginable incluye tres métodos:

Nombre de método	**Descripción**
int getNumberOfPages ()	Devuelve el número de páginas en el documento.
PageFormat getPageFormat (int pagina)	Devuelve el PageFormat de determinada página cuyo número se especifica según pagina.
Printable getPrintable (int pagina)	Devuelve el Printable e instancia responsable de renderizar la página cuyo número se especifica según pagina.

- Interfaz Printable

El interfaz Printable destaca un método y dos valores:

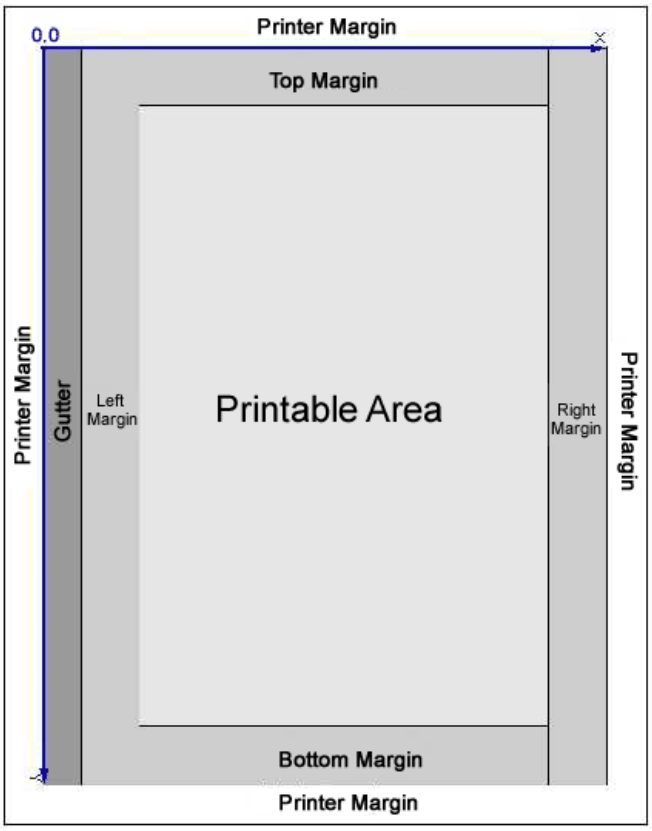
Nombre	**Tipo**	**Descripción**
int print (Graphics, graphics, PageFormat pf, int pagina)	Método	Requiere que a través de Graphics, utilizando el PageFormat dado, renderice la página especificada mediante pagina.
NO_SUCH_PAGE	Valor	Esta es un constante. Se devuelve este valor para indicar que no hay más páginas para imprimir.
PAGE_EXISTS	Valor	El método print() devuelve PAGE_EXISTS. Esto indica que la página pasada como un parámetro a print () ha sido renderizada y por ende ésta existe.

Cada pintor de página debe implementar el interfaz Printable. Ya que hay sólo un método a implementar, creando pintores de página puede pensarse que está todo resuelto. Sin embargo, debemos tener presente que el código que escribamos debe ser capaz de dar cualquier página en una o cualquiera determinada secuencia.

Hay tres parámetros para imprimir, incluyendo Graphics, que es la misma clase que se usa para dibujar en la pantalla. Ya que la clase Graphics implementa el interfaz PrinterGraphics, podemos obtener el PrinterJob que instanció este trabajo de impresión. Si la disposición de página es compleja y se requiere de rasgos de dibujo

avanzados, podemos castear el parámetro Graphics a un objeto Graphics2D.

Antes que se comience a usar el objeto Graphics, debemos notar que las coordenadas no son trasladadas a la esquina superior izquierda del área imprimible. Ve la siguiente Figura 3 para encontrar la posición de origen que falta.



Coordenadas de origen del área publicable

La coordenada de origen (0, 0) aparece en la esquina superior izquierda en los márgenes de impresora. Para imprimir un rectángulo de 1 pulgada cuadrada desde la esquina superior izquierda de margen (del área que es imprimible) deberíamos escribir el siguiente código:

```
public int print (Graphics g, PageFormat pF, int pagina){

        Graphics2D g2d = (Graphics2D) g;

        Rectangle2D.Double rect = new Rectangle2D.Double ();

        rect.setRect (pF.getImageableX () + 72,

                        pF.getImageableY () + 72,  72, 72);

        Graphics2D.draw (rect);

        return (PAGE_EXISTS);

}
```

Del ejemplo anterior, nosotros vemos que manualmente debemos trasladar el origen del rectángulo de modo que esto imprima en lo alto del área imprimible. Para simplificar el código, nosotros podríamos trasladar las coordenadas una vez y emplear (0, 0) como el origen del área imprimible. Modificando el ejemplo anterior, tenemos:

```
public int print (Graphics g, PageFormat pF, int pagina){

        Graphics2D g2d = (Graphics2D) g;

        g2d.translate(pF.getImageableX(), pF.getImageableY());
        Rectangle2D.Double rect = new Rectangle2D.Double ();
        rect.setRect (72, 72,  72, 72);
        Graphics2D.draw (rect);
        return (PAGE_EXISTS);
}
```

Usando el método translate, podemos trasladar las coordenadas y poner nuestro punto de origen (0, 0) en lo alto del área imprimible. Desde este punto, nuestro código será simplificado.

- Interfaz PrinterGraphics

El interfaz PrinterGraphics consiste en un método:

Nombre de método	**Descripción**
PrinterJob getPrinterJob ()	Devuelve el PrinterJob para un determinado renderizamiento y además este método es implementado por la clase Graphics.

- La Clase Paper

Ocho métodos constituyen la clase Paper:

Nombre de método	**Descripción**
double getHeight ()	Este método devuelve la altura física de la página en puntos (1 pulgada = 72 puntos). Por ejemplo, si imprimimos en una página de tamaño de carta, el valor de vuelta será 792 puntos, o 11 pulgadas.
double getImageableHeight ()	Este método devuelve la altura imprimible de una página. La altura imprimible es la altura del área de impresión que se puede utilizar.

double getImageableWidth ()	Este método devuelve la anchura imprimible de una página. La anchura imprimible es la anchura del área de impresión que se puede utilizar.
double getImageableX ()	Este método devuelve el origen X del área imprimible. Ya que no hay ningún apoyo de margen (como por ejemplo para encuadernar), el valor de vuelta representa el margen izquierdo.
double getImageableY ()	Este método devuelve el origen Y del área imprimible. El valor devuelto de este método es equivalente al margen superior.
double getWidth ()	Este método devuelve la anchura física de la página en puntos. Si se imprime sobre un papel tamaño de carta, la anchura es 8.5 pulgadas, o 612 puntos.
Void setImageableArea (double x, double y, la double ancho, double alto)	Este método pone el área imprimible y especifica los márgenes sobre la página. En realidad, el API no proporciona ningún método de poner los márgenes explícitamente; debemos nosotros calcularlos.
Void setSize (double ancho, double alto)	Este método fija el tamaño físico de la página. Para definir una hoja 8.5 por 11 pulgadas, deberíamos disponer 612 y 792 puntos respectivamente. Note que el tamaño por defecto es CARTA.

Antes que sigamos adelante, recordemos que **la clase Paper define las características físicas de la página**. **La clase PageFormat representa las características de toda la página**, como la orientación de página, el tamaño, y el tipo de papel. Esta clase siempre es pasada como un parámetro para la Interfaz Printable, mediante el método print(). Use Paper para obtener la posición del área imprimible, tamaño, y orientación de página con una matriz de transformación.

- La Clase PageFormat

El PageFormat consiste en 12 métodos:

Nombre de método	**Descripción**
double getHeight ()	Este método devuelve la altura física de la página en puntos (1 pulgada = 72 puntos). Si su página mide 8.5 en 11 pulgadas, entonces el valor de vuelta será 792 puntos, o 11 pulgadas.
double getImageableHeight ()	Este método devuelve la altura imageable de la página, que es la altura de la impresión el área sobre la cual se puede dibujar.

double getImageableWidth ()	Este método devuelve la anchura imageable de la página - la anchura de la impresión el área sobre la cual se puede dibujar.
double getImageableX ()	Este método devuelve el origen X del área imageable.
double getImageableY ()	Este método devuelve el origen Y del área imageable.
double getWidth ()	Este método devuelve la anchura física de la página en puntos. Si se imprime sobre el papel de carta, la anchura es 8.5 pulgadas, o 612 puntos.
double getHeight ()	Este método devuelve la altura física de la página en puntos. Por ejemplo, el papel de carta es de 11 pulgadas en la altura, o 792 puntos.
double [] getMatrix ()	Este método devuelve una matriz de transformación que traduce el espacio de usuario en la orientación de página solicitada. El valor de vuelta está en el formato requerido por el constructor AffineTransform.
Int getOrientation ()	Este método devuelve la orientación de la página como PORTRAIT O LANDSCAPE.
Void setOrientation (int la orientación)	Este método pone la orientación de la página, usando las constantes PORTRAIT y LANDSCAPE.
Paper getPaper ()	Este método devuelve el objeto Paper asociado con el formato de página. Refiérase a la sección anterior para una descripción de la clase Paper.
Void setPaper (Paper papel)	Este método pone el objeto Paper que será usado por la clase PageFormat. PageFormat debe tener el acceso a las características de página físicas para completar esta tarea.

Esto concluye la descripción de las clases de página. A continuación veamos la clase PrinterJob.

- La Clase PrinterJob

La clase PrinterJob controla el proceso de impresión. Esto puede tanto instanciar como controlar un trabajo de impresión. Veamos los métodos de esta clase:

Nombre de método	**Descripción**

abstract void cancel()	Este método cancela el trabajo de impresión actual. Usted puede validar la cancelación con el método isCancel.
abstract boolean isCancelled()	Estas método retorna verdadero si el trabajo es cancelado.
PageFormat defaultPage()	Este método devuelve el formato de página al PrinterJob.
abstract PageFormat defaultPage(PageFormat page)	Este método copia el PageFormat pasado en parámetros y modifica la copia para dejar PageFormat por defecto.
abstract int getCopies()	Este método retorna el número de copias que el trabajo de impresión imprimirá.
abstract void setCopies(int copies)	Este método pone el número de copias que el trabajo imprimirá. Note que si se muestra un cuadro de diálogo de impresión, los usuarios pueden cambiar el número de copias.
abstract String getJobName()	Este método devuelve el nombre del trabajo.
static PrinterJob getPrinterJob()	Este método crea y retorna un nuevo PrinterJob.
abstract String getUserName()	Este método devuelve el nombre de usuario asociado con el trabajo de impresión.
abstract PageFormat pageDialog(PageFormat page)	Este método muestra un diálogo que permite al usuario modificar el PageFormat. El PageFormat, pasado en parámetros, pone los campos del diálogo. Si el usuario cancela el diálogo, entonces el PageFormat original será devuelto. Pero si el usuario acepta los parámetros, entonces se crea un nuevo PageFormat siendo este el que se retorna. Ya que esto no mostrará los mismos parámetros sobre todos los sistemas de operaciones, se debe ser cuidadoso usando el pageDialog.
abstract void setPageable(Pageable document)	Este método pregunta por los documentos a imprimir, para obtener el número total de páginas. El Paginable también devolverá el PageFormat y el objeto Printable para cada página..
abstract void setPrintable(Printable painter)	Este método pone el objeto Pintor que dará las páginas para ser impresas. Un objeto Pintor es un objeto que implementa la clase Printable y su método print().
abstract void setPrintable(Printable painter, PageFormat format)	Este método completa las mismas tareas que abstract void setPrintable(Printable painter), excepto que nosotros suministramos el PageFormat que el Pintor usará..
abstract void print()	Este método imprime el documento. Este en realidad llama al método print() del Pintor antes asignado a este trabajo de impresión.
abstract void setJobName(String jobName)	Este método pone el nombre del trabajo de impresión.

abstract boolean printDialog()	Este método muestra un cuadro de diálogo de impresión que permite al usuario cambiar los parámetros de impresión. Note que el resultado de esta interacción no será devuelto a su programa. En cambio, será pasado al sistema operativo.
abstract PageFormat validatePage(PageFormat page)	Este método validará el PageFormat pasado en parámetros. Si la impresora no puede usar el PageFormat que suministramos, entonces uno nuevo que posee por defecto la impresora será devuelto.

La clase Book

Siete métodos constituyen la clase Book:

Nombre de Método	**Descripción**
void append(Printable painter, PageFormat page)	Este método añade una página *al Libro*. *El pintor y el Page*Format para aquella página son pasados en parámetros.
void append(Printable painter, PageFormat page, int numPages)	Este método completa las mismas tareas que el void append(Printable painter, PageFormat page)), pero nosotros especificamos el número de páginas.
int getNumberOfPages()	Este método devuelve el número de páginas actualmente en el Libro.
PageFormat getPageFormat(int pagina)	Este método devuelve el objeto PageFormat para una página dada.
Printable getPrintable(int pagina)	Este método devuelve al pintor para una página dada.
void setPage(int pagina, Printable painter, PageFormat page)	Este método pone al pintor y el PageFormat para una página dada en el libro.

Definiendo los pasos para imprimir

Debemos seguir los siguientes pasos si deseamos imprimir en Java:

a) Primero se debe crear un objeto PrinterJob:

PrinterJob printJob = PrinterJob.getPrinterJob ();

b) Después, usando el método setPrintable () del PrinterJob, asignar el objeto Pintor al PrinterJob. Note que un objeto Pintor es el que implementa el interfaz Publicable.

printJob.setPrintable (Pintor);

O podríamos poner el PageFormat (formato de página) con el pintor:

printJob.setPrintable (Pintor, FormatoDePagina);

c) Finalmente, el objeto Painter debe implementar el método imprimir():

public int print(Graphics g, PageFormat pF, int pagina);

Aquí, el primer parámetro es utilizado para renderizar la pagina, o dicho de otra forma para crear el dibujo sobre la pagina que queremos imprimir; el pageFormat es el formato que será usado para la página corriente; y el último parámetro es el número de página a ser renderizado.

Esto es todo lo que hay que hacer, fácil no???

Imprimiendo nuestra primera pagina

Aunque Printable puede imprimir documentos simples, este destaca varias limitaciones, el principal es que todas las páginas deben compartir el mismo formato. El modelo Pageable, ofrece mucho más flexibilidad en este caso. Usadando Pageable en conjunto con la clase Book podemos crear documentos de múltiples páginas, con cada página formateada de manera diferente.

Examinenos un poco los pasos requeridos para imprimir con el modelo Printable :

1. Cree un objeto PrinterJob. Este objeto controla el proceso de impresión mostrando la página y diálogos de impresión, e iniciando la acción de impresión.

2. Muestre los diálogos apropiados, de impresión o de página.

3. Cree una clase que implemente la interfaz Printable, mediante el método print().

4. Valide el número de página para ser renderizado (o dibujado).

5. Dibuje su página usando el parámetro graphics.

6. Si la página se dibuja, devuelva el valor de PAGE_EXISTS; en caso contrario, devuelva el valor de NO_SUCH_PAGE.

Veamos el siguiente código de ejemplo:

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.Rectangle2D;
import java.awt.print.*;

public class Ejemplo1 implements Printable{

public Ejemplo1() {
        super();
PrinterJob printJob = PrinterJob.getPrinterJob();
printJob.setPrintable(this);
if (printJob.printDialog()) {
try {
printJob.print();
        } catch (Exception PrinterException) {
PrinterException.printStackTrace();
}
}
}
```

```
public int print(Graphics g, PageFormat pf, int pagina){

Graphics2D g2d;

if (pagina==0) {

g2d = (Graphics2D)g;

g2d.setColor(Color.black);

g2d.translate(pf.getImageableX(),pf.getImageableY());


Rectangle2D.Double rect = new Rectangle2D.Double(

0,0,pf.getImageableWidth(),pf.getImageableHeight());


g2d.draw(rect);

return (PAGE_EXISTS);

} else {

return (NO_SUCH_PAGE);

}

}


}
```

Use Pageables y Libros para imprimir

Como usted aprendió del ejemplo anterior, imprimir con Printable es un proceso directo, pero no ofrece mucha flexibilidad cuando se requieren documentos más complejos a manejar. Para estos tipos de tareas, las clases Pageable y Book entran en acción. La clase Book permite escoger si la página de cada Book usará un pintor de página diferente o el mismo pintor de página que otras páginas.

Este ejemplo imprimirá un documento de dos páginas que usa un pintor para la portada y otro pintor para el documento en sí. La portada imprimirá sobre una hoja en forma vertical, mientras la segunda página será puesta en forma horizontal. Note que en este código, cada pintor imprime una página; no se valida el parámetro de página. Para usar a un pintor para imprimir más que una página, considere el siguiente ejemplo:

```
//Ejemplo2.java

import java.awt.print.*;

public class Ejemplo2 {


public Ejemplo2() {

        super();

PrinterJob printJob = PrinterJob.getPrinterJob();

        Book libro = new Book();

        libro.append(new Portada(),printJob.defaultPage());
```

```
        PageFormat documentoPF = new PageFormat();
documentoPF.setOrientation(PageFormat.LANDSCAPE);
        libro.append(new Documento(),documentoPF);
printJob.setPageable(libro);
        if (printJob.printDialog()) {
try {
printJob.print();
} catch (Exception PrinterException) {
PrinterException.printStackTrace();
}
}
}

}
//Portada.java
import java.awt.Color;
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.Rectangle2D;
import java.awt.print.*;
public class Portada implements Printable{
public int print(Graphics g, PageFormat pf, int page){
final double PULGADA = 72;
Graphics2D g2d = (Graphics2D)g;
g2d.translate(pf.getImageableX(),pf.getImageableY());
g2d.setPaint(Color.black);
Rectangle2D.Double borde = new Rectangle2D.Double(
0,0,pf.getImageableWidth(),pf.getImageableHeight());
g2d.draw(borde);
String Titulo = "Imprimiendo con Java";
Font fuenteTitulo = new Font("helvetica",Font.BOLD,36);
g2d.setFont(fuenteTitulo);
FontMetrics medidaFuente = g2d.getFontMetrics();
```

```
double tituloX = (pf.getImageableWidth()/2) -

(medidaFuente.stringWidth(Titulo)/2);

double tituloY = 3 * PULGADA;

g2d.drawString(Titulo,(int)tituloX, (int)tituloY);

return (PAGE_EXISTS);

}

}

//Documento.java

import java.awt.BasicStroke;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.geom.Rectangle2D;

import java.awt.print.*;

public class Documento implements Printable{


public int print(Graphics g,PageFormat pf, int page){

final int PULGADA = 72;

Graphics2D g2d = (Graphics2D)g;

g2d.translate(pf.getImageableX(),pf.getImageableY());

g2d.setColor(Color.black);

g2d.setStroke(new BasicStroke(12));

Rectangle2D.Double borde = new Rectangle2D.Double(

0,0,pf.getImageableWidth(),

pf.getImageableHeight());

g2d.draw(borde);

g2d.drawString("El contenido",PULGADA,PULGADA);

return (PAGE_EXISTS);

}

}
```

En este otro ejemplo, se añade una tercera página utilizando el mismo pintor que se uso en la segunda. (Utilizamos la misma clase Portada descrita anteriormente)

```
//Ejemplo3.java

import java.awt.print.Book;

import java.awt.print.PageFormat;
```

```
import java.awt.print.PrinterJob;


public class Ejemplo3 {


public Ejemplo3() {

super();

PrinterJob printJob = PrinterJob.getPrinterJob();

Book libro = new Book();

libro.append(new Portada(),printJob.defaultPage());

PageFormat documentoPF = new PageFormat();

documentoPF.setOrientation(PageFormat.LANDSCAPE);

libro.append(new Documento2(),documentoPF);

libro.append(new Documento2(),documentoPF);

printJob.setPageable(libro);

if (printJob.printDialog()) {

try {

printJob.print();

} catch (Exception PrinterException) {

PrinterException.printStackTrace();

}

}

}


}

//Documento2.java

import java.awt.BasicStroke;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.geom.Rectangle2D;

import java.awt.print.PageFormat;

import java.awt.print.Printable;


public class Documento2 implements Printable{

public int print(Graphics g,PageFormat pf, int page){
```

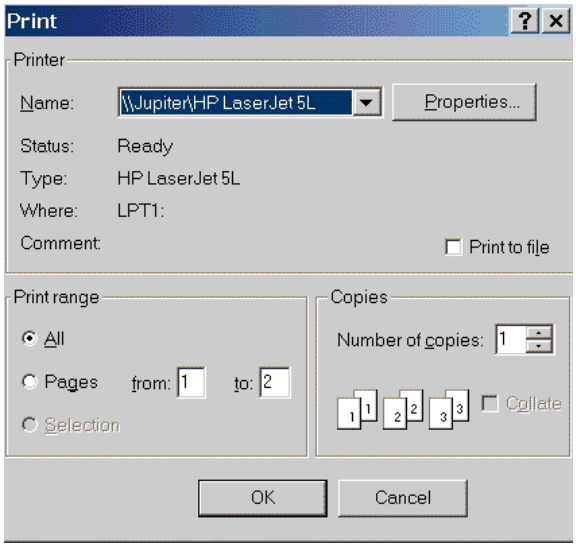
```
final int PULGADA = 72;

Graphics2D g2d = (Graphics2D)g;

g2d.translate(pf.getImageableX(),pf.getImageableY());

g2d.setColor(Color.black);

g2d.setStroke(new BasicStroke(12));

Rectangle2D.Double borde = new Rectangle2D.Double(

0,0,pf.getImageableWidth(),

pf.getImageableHeight());

g2d.draw(borde);


if (page == 1) {

g2d.drawString("Contenido de la pagina " + page,

PULGADA,PULGADA);

return (PAGE_EXISTS);

} else if (page == 2){

g2d.drawString("Contenido de la pagina " + page,

PULGADA,PULGADA);

return (PAGE_EXISTS);

}

return (NO_SUCH_PAGE);

}

}
```

Ahora que sabemos renderizar (o dibujar) páginas que usan la API de impresión, podría ser útil presentar un diálogo con el cual el usuario puede escoger la impresora a utilizar. También podríamos querer dar al usuario una oportunidad de cambiar el formato de página y márgenes.

La API de impresión ofrece dos ventanas de diálogo, el diálogo de impresora y el diálogo de página.

Usando diálogos de impresión

Utilización diálogos de impresión damos una interfaz mas amigable al usuario, para determinadas tareas las cuales el desea realizar al imprimir. Veamos la siguiente figura:



Diálogo de impresión

Con el cuadro de diálogo de impresión, el usuario puede seleccionar una impresora y modificar su configuración. El usuario también puede poner el número de copias a imprimir y seleccionar de una gama de página.

Usando este diálogo, se transfiere el control del trabajo de impresión al usuario. El cuadro de diálogo proporciona el único lugar donde el usuario puede decidir seguir con el trabajo de impresión o cancelarlo.

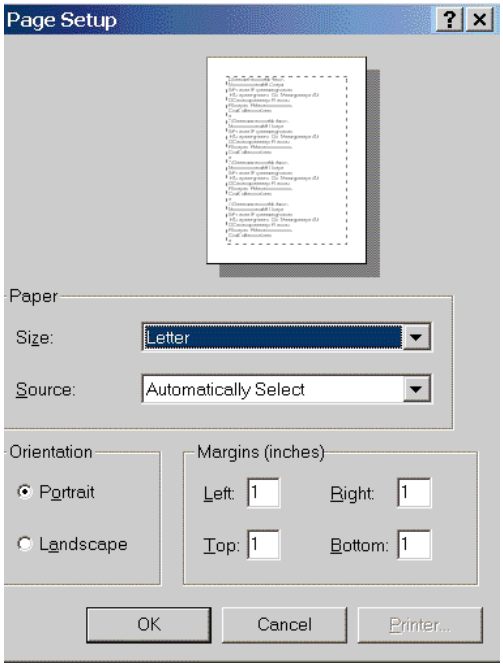
Veamos un pequeño extracto de código que muestra el diálogo de impresión:

```
if (printJob.printDialog()) {
     try {
       printJob.print();
     }
     catch (Exception PrintException) {
       PrintException.printStackTrace();
     }
}
```

El método printDialog(), que es parte de la clase PrinterJob, muestra el diálogo de impresión al usuario. Este método devuelve un valor Booleano. Si el usuario selecciona el botón imprimir, printDialog () devolverá verdadero; si el usuario selecciona el botón cancelar, el método devuelve falso, y el trabajo de impresión no procederá.

El método printDialog() sólo actúa con el programa devolviendo el valor Booleano (es decir si el usuario aceptó o no imprimir). Ninguno de los parámetros del cuadro de Diálogo (que modificque el usuario) afecta al objecto PrinterJob.

El segundo diálogo es el **diálogo de página**. Con este diálogo, el usuario puede cambiar el sistema de página. Vea la figura abajo:



Diálogo de página

Usando este diálogo, el usuario puede escoger todos los parámetros del objeto PageFormat, incluyendo el tamaño de papel, la fuente de papel, la orientación de la página, y los márgenes.

El siguiente extracto de código muestra como invocar al diálogo de página:

```
 PageFormat documentPageFormat = new PageFormat ();
documentPageFormat = printJob.pageDialog (documentPageFormat);
book.append (new Document (), documentPageFormat);
```

Si el usuario selecciona el botón OK, el método pageDialog() devolverá una copia del objeto PageFormat que refleja la elección del usuario. Si el usuario selecciona el botón Cancelar, entonces el método devuelve una copia inalterada de PageFormat.

Ahora se viene una parte compleja del manejo de impresión. La impresión de textos

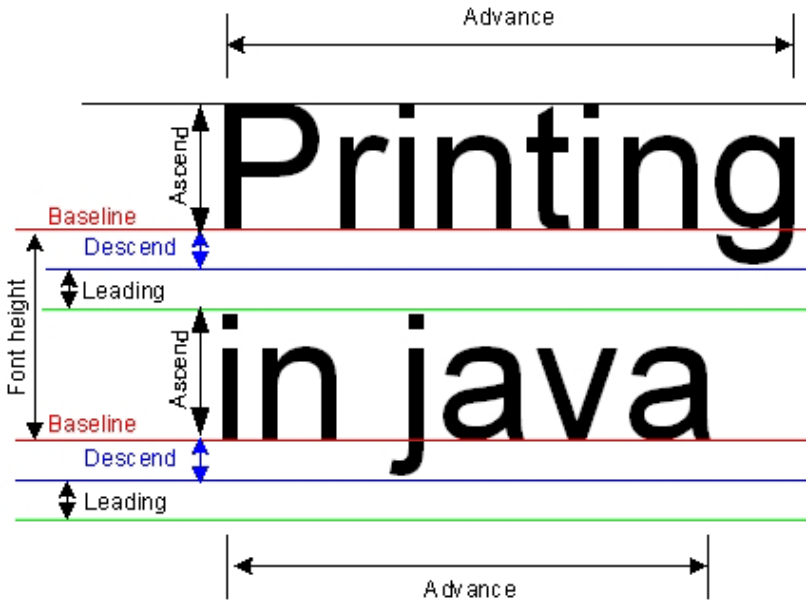
Manejo de Texto y Fuentes

La renderización del texto probablemente representa el aspecto más complejo de impresión. Mucho tiempo ha pasado de los días de las impresoras de matriz de punto, cuando sólo tuvimos que saber el número de caracteres por pulgada y la anchura de la página en caracteres. La justificación completa de un párrafo era simple; solamente añadimos espacios entre cada palabra hasta que el texto quedara alineado con el margen derecho. Para alinear el texto verticalmente, sólo enviamos un código a la impresora y comenzamos a imprimir la siguiente línea. Sin embargo, como ha cambiado esto para hoy en día. Usando la tecnología de hoy para imprimir dos líneas de texto, un encima del otro, exige mucho más conocimiento, incluyendo como usar fuentes.

a) Fuentes

Primero debemos entender la estructura de una fuente. La figura (abajo) ilustra como las fuentes son presentadas. Los caracteres alinean a lo largo de una línea llamada **línea de fondo**, que es el punto vertical de referencia para colocar una fuente. El ascendiente es la distancia entre la línea de fondo y la cima del carácter más alto en un String. El espacio entre la línea de fondo y la parte más baja de una String es **el descendiente**. La distancia vertical entre dos caracteres es el leading, y la suma del Descendente con el leading es la altura de la fuente.

Finalmente, llamamos **el avance** a la longitud de una String. Todos estos parámetros son dependiente de la fuente, significando esto que el texto que utiliza una fuente Arial no ocupará el mismo espacio físico que el mismo texto dado usando una fuente Lucida.



El proceso de colocar fuentes se diferencia de colocar una figura(un número) estándar geométrica. La parte superior izquierda de un rectángulo es donde se pone el origen; la línea de fondo sobre el Eje de ordenadas y el primer carácter sobre el Eje de abscisas marca el origen de una fuente. Observa la siguiente figura (el Número) 4 para ver la posición del origen de fuente.

La figura(El número) 4. Origen de fuente

Dos estilos de fuentes están disponibles:

Serif fuentes se destacan por ser líneas cortas que contienen un ángulo en las partes superiores e inferiores de una letra. Time new roman es un ejemplo de una fuente **Serif.**

Sans serif fuentes no tienen ningún elemento decorativo. La fuente Arial es una fuente **sans serif**.

Cada uno de estos estilos de fuente puede ser **monoespaciado o proporcional**.

Una fuente **monoespaciada** es una fuente en la cual todos los carácteres tienen la misma anchura. Viejas impresoras, como las de matrices, usan fuentes monoespaciadas, como base de texto.

Cada carácter en una fuente **proporcional** posee una ancho diferente.

Las fuentes proporcionales se dieron a conocer cuando las impresoras de láser salieron al mercado. Los primeros ordenadores comerciales para usar fuentes proporcionales tanto sobre la pantalla como sobre la impresora eran Apple y Macintosh.

b) Tener acceso a fuentes

Como lo dice su lema: â€œdiseñado para ser escrito una vez y controlar en todas partesâ€ , los creadores de java tuvieron que producir una familia de fuente por defecto para las diferentes plataformas. Java ofrece ocho fuentes por defecto: Serif, Sans Serif, Dialog, Dialog Input, Lucida Sans, Lucida Sans Typewriter, Lucida Bright, y Monospace.

c) Clases de fuente relacionadas

El API Graphics suministra muchas clases que alivian la tarea de manipular fuentes. Las definiciones de algunas clases más útiles se muestran en la siguiente tabla:

Nombre	Tipo	Descripción
Font	Clase	La clase de Fuente representa una instancia de un tipo de fuente. Use esta clase para crear una nueva fuente basada en las fuentes disponibles sobre el sistema objetivo.
FontMetrics	Clase (Abstracta)	Esta clase contiene la información sobre una fuente y el dispositivo de interpretación. En FontMetrics se destacan muchas utilidades, una de cual obtiene la anchura de un string. Note que esta clase es abstracta; para adquirir una instancia, se debe llamar a Graphics.getFontMetrics (). Use FontMetrics para aprender la información básica métrica de una fuente, como su descendiente, ascendente, el leading, o el avance.
FontRenderContext	Clase	Esta clase proporciona la información sobre una fuente relacionada con el dispositivo de interpretación y las reglas requeridas para dar tal fuente. Las reglas definen la calidad de la interpretación.

No es fácil renderizar los párrafos de textos. La escritura de un algoritmo para justificar el texto que tiene una fuente proporcional es una tarea laboriosa. La agregación del apoyo a caracteres internacionales sólo añade mas complejidad. Es por eso que usaremos el TextLayout y las clases LineBreakMeasurer para poder justificar los párrafos

d) Usando TextLayout

TextLayout ofrece mucha funcionalidad en la interpretación del texto de alta calidad. Esta clase puede dar el texto bidireccional como el texto japonés, donde las figuras alinean de derecho hhacia abajoen vez del estilo norteamericano, que fluye de la izquierda a la derecha.

Con TextLayout podemos tener nuestros párrafos, pero este no trabaja solo. Para arreglar el texto dentro de una anchura especificada, se necesita la ayuda de la clase LineBreakMeasurer. LineBreakMeasurer se adecua a un string para encajar una anchura predefinida. Ya que esta es una clase multilingüe, esta conoce exactamente donde romper una línea de texto según las reglas de cada lengua. La clase LineBreakMeasurer también no trabaja sola. Necesita la información de la clase FontRenderContext, que, como su función principal, devuelve el métrico de fuente exacto. Para medir el texto con eficacia, FontRenderContext debe saber las formas de renderizar el texto para el dispositivo dado junto con el tipo de fuente usado.

Falta introducir otra clase: **AttributedString**. Esta clase es sumamente provechosa cuando se quiere encajar la información de atributo dentro de un string.

Por ejemplo, digamos que nosotros tengamos la oración siguiente:

Esta es una **Negrita** letra.

Si se imprimiera este string sin la clase AttributedString, se escribiría algo como esto:

```
Font normalFont = new Font ("serif", Font.PLAIN, 12);
  Font boldFont = new Font ("serif", Font.BOLD, 12);
  g2.setFont (normalFont);
  g2.drawString ("Esta es una");
  g2.setFont (boldFont);
  g2.drawString ("Negrita");
  g2.setFont (normalFont);
  g2.drawString ("letra", 72, 72);
```

Usando **AttributedString:**

```
AttributedString attributedString = new AttributedString ("Esta en una Negrita letra");
 attributedString.addAttribute (TextAttribute.WEIGHT,
 TextAttribute.WEIGHT_BOLD, 11, 14);
 g2.drawString (attributeString.getIterator (), 72, 72);
```

El siguiente ejemplo, imprime un documento de varias lineas de texto, donde el texto esta en forma justificada

```
//DocumentoConParrafos.java

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.font.FontRenderContext;
import java.awt.font.LineBreakMeasurer;
import java.awt.font.TextAttribute;
import java.awt.font.TextLayout;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import java.awt.print.PageFormat;
import java.awt.print.Printable;
import java.text.AttributedString;
import java.util.Vector;
import javax.swing.text.html.HTML.Attribute;

public class DocumentoConParrafos implements Printable{
public int print(Graphics g,PageFormat pf, int page){

final int PULGADA = 72;
Graphics2D g2d = (Graphics2D)g;
g2d.translate(pf.getImageableX(),pf.getImageableY());
g2d.setColor(Color.black);
g2d.setStroke(new BasicStroke(4));
Rectangle2D.Double borde = new Rectangle2D.Double(
0,0,pf.getImageableWidth(),
pf.getImageableHeight());
g2d.draw(borde);
String texto = new String();

texto += "A lo largo de este Tutorial, se ha aprendido ";
texto += "mediante las clases EjemploX a imprimir un ";
```

```
texto += "documento de diferentes formas. Lo mas dificil ";
texto += "es la impresion de texto, ya que se debe tener ";
texto += "mas conocimiento en lo referente a fuentes. ";
texto += "Se añora esos tiempos donde no debiamos, ";
texto += "preocuparnos de estas cosas. Sin embargo, ";
texto += "guardo esperanza que esta forma de imprimir ";
texto += "cambiara y sera mucho mas facil... ";
Point2D.Double punto = new Point2D.Double(
0.25 * PULGADA, 0.25 * PULGADA);

double ancho = 8 * PULGADA;

AttributedString parrafo = new AttributedString(texto);
parrafo.addAttribute(TextAttribute.FONT,
new Font("Serif", Font.PLAIN, 12));
LineBreakMeasurer lineaBreaker = new LineBreakMeasurer(
parrafo.getIterator(),
new FontRenderContext(null,true,true));

TextLayout Campo;
TextLayout justCampo;

Vector lineas = new Vector();
while ( (Campo = lineaBreaker.nextLayout((float)ancho))
!= null) {
lineas.add(Campo);
}
for (int i = 0; i < lineas.size(); i++) {
Campo = (TextLayout)lineas.get(i);

if (i != lineas.size()-1) {
                                justCampo = Campo.getJustifiedLayout((float)ancho);

} else {
justCampo = Campo;
```

}

punto.y += justCampo.getAscent();

justCampo.draw(g2d,(float)punto.x,(float)punto.y);

punto.y += justCampo.getDescent() +

justCampo.getLeading();

}

return (PAGE_EXISTS);

}

}

Impresión de imágenes

La impresión de una imagen estal cual como se carga un objeto de Imagen, invocando al método Graphics2D.drawImage().

El proceso de imprimir una imagen es dividido en tres pasos fáciles.

1. Cree un URL que indicará la imagen que quieres imprimir.

2. Cargue la imagen que usa una clase MediaTracker. Con esta clase, usted puede cargar a GIF, JPEG, y archivos PNG. Si se desea cargar otros tipos de imagen, use la API de Imágenes avanzadas

3. Dibuje la imagen usando el método drawImage () de la clase Graphics2D. drawImage () necesita seis parámetros. El primero es el objeto Imagen a imprimir; el segundo y tercer parámetro son coordenadas para el borde superior izquierdo de la imagen la cual le da la posición a esta; el cuarto y quinto parámetros especifican el ancho y alto de la imagen. El último parámetro es para el ImageObserver.

El ejemplo siguiente ilustra como imprimir una imagen:

Para el objetivo de este ejemplo, el DocumentoConImagen.java amplía la clase Componente. Todas las clases Componentes ponen en práctica el interfaz ImageObserver.

//DocumentoConImagen.java

import java.awt.BasicStroke;

import java.awt.Color;

import java.awt.Component;

```
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.MediaTracker;
import java.awt.Toolkit;
import java.awt.geom.Rectangle2D;
import java.awt.print.PageFormat;
import java.awt.print.Printable;
import java.net.MalformedURLException;
import java.net.URL;

public class DocumentoConImagen extends Component
implements Printable{

public int print(Graphics g,PageFormat pf, int page){

final int PULGADA = 72;
Graphics2D g2d = (Graphics2D)g;
g2d.translate(pf.getImageableX(),pf.getImageableY());
g2d.setColor(Color.black);
g2d.setStroke(new BasicStroke(12));
Rectangle2D.Double borde = new Rectangle2D.Double(
0,0,pf.getImageableWidth(),
pf.getImageableHeight());
g2d.draw(borde);

MediaTracker mt = new MediaTracker(this);
URL imagenURL = null;
try {
imagenURL = new URL("E:/logo_eclipse.gif");
} catch (MalformedURLException e) {
e.printStackTrace();
}
                    Image imagen = Toolkit.getDefaultToolkit().getImage(imagenURL);
                    mt.addImage(imagen,0);
```

```
try {
mt.waitForID(0);
} catch (InterruptedException e) {
        }
g2d.drawImage(imagen, (int)(0.25 * PULGADA),
(int)(0.25 * PULGADA),(int)(8.5 * PULGADA),
(int)(6 * PULGADA),this);
return (PAGE_EXISTS);
}
}
```

Referencias

http://www.javaworld.com/javaworld/jw-10-2000/jw-1020-print.html

http://www.javaworld.com/javaworld/jw-12-2000/jw-1201-print.html

Puedes opinar sobre este tutorial aquí

Recuerda

que el personal de Autentia te regala la mayoría del conocimiento aquí compartido (Ver todos los tutoriales)

¿Nos vas a tener en cuenta cuando necesites consultoría o formación en tu empresa?
¿Vas a ser tan generoso con nosotros como lo tratamos de ser con vosotros?

info@autentia.com

Somos pocos, somos buenos, estamos motivados y nos gusta lo que hacemos .....
Autentia = Soporte a Desarrollo & Formación

Nuevo servicio de notificaciones

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales, inserta tu dirección de correo en el siguiente formulario.

Subscribirse a Novedades	
e-mail	
	Enviar

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto **Descripción**

Nota: Los tutoriales mostrados en este Web tienen como objetivo la difusión del conocimiento.

Los contenidos y comentarios de los tutoriales son responsabilidad de sus respectivos autores.

En algún caso se puede hacer referencia a marcas o nombres cuya propiedad y derechos es de sus respectivos dueños. Si algún afectado desea que incorporemos alguna reseña específica, no tiene más que solicitarlo.

Si alguien encuentra algún problema con la información publicada en este Web, rogamos que informe al administrador rcanales@adictosaltrabajo.com para su resolución.