IES de Teis
Avda. de Galicia, 101
36216 – Vigo
Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es

GESTORES DE PRESENTACION

INDICE

1. Introducción....2
1.1 Introducción a los Layouts Managers:....3
1.2 Introducción a Window:....3
2. Concepto de Layouts Managers:....3
2.1 Ventajas y Desventajas de los Layout Managers:....4
2.2 Contenedores y Componentes....5
2.3 Reglas Generales para el uso de Controladores de Disposición....6
3. Flow Layout....7
4. BorderLayout....9
5. CardLayout....10
6. GridLayout:....12
7. GridBagLayout:....13
7.1 GridBagLayout a fondo:....14
7.2 Ejemplo GridBagLayout :....16
8. Ventanas, Marcos y Cuadros de Diálogo:....18
8.1 Clase Window:....18
8.2 Clase Frame:....19
8.3 Clase Dialog:....19
8.4 Clase FileDialog....20
8.5 Ejemplo práctico:....21
9. Menús:....23
9.1 Construcción de un menú....23

Capitulo 1: Gestores de presentación en el entorno . Ventanas, marcos, cuadros de diálogo e menús.

1.Introducción

La portabilidad de Java a distintas plataformas y distintos sistemas operativos necesita fexibilidad a la hora de situar los Components (Buttons, Canvas, TextAreas, etc.) en un contenedor (Window, Panel, …). Un Layout Manager es un objeto que controla cómo los componentes se sitúan en un contenedor.

Un Controlador de disposición es un objeto que controla el tamaño y posición de los componentes de un contenedor. Los controladores de disposición se adhieren al interface LayoutManager. Por defecto, todos los objetos Container tiene un objeto LayoutManager que controla su distribución. Para los objetos de la clase Panel, el controlador de disposición por defecto es un ejemplar de la clase FlowLayout. Para los objetos de la clase Window, el controlador de disposición por defecto es un ejemplar de la clase BorderLayout.

En la primera parte de este tema vamos a abordar cómo utilizar los controladores de disposición que proporciona el AWT. La segunda parte del tema está orientada a aquellas aplicaciones gráficas que no son applets y que requieren de una clase especial llamada **Window** y de dos extensiones suyas que se usan con mayor frecuencia que Window: **Frame** y **Dialog**, que en español se llaman **marco** y **cuadro de diálogo** respectivamente.

1.1Introducción a los Layouts Managers:

Los layout managers son uno de los conceptos más útiles que podemos encontrar en Java. Gracias a ellos podremos organizar todos los componentes de nuestra interfaz gráfica de modo que sea más sencillo añadirlos, eliminarlos o recolocar su posición. Los layout managers automatizan una gran cantidad de trabajo y eliminan al programador la necesidad de realizar tediosas tareas de control del interfaz.

Una traducción libre del término sería manejador de contenido y en realidad layout manager eso es lo que es. No es más que un delegado que se encarga de organizar los componentes que forman parte de un contenedor como por ejemplo pueda ser una ventana. El es el encargado de decidir en que posiciones se renderizarán los componentes, que tamaño tendrán, que porción del contenedor abarcarán, etc... Todo esto se realiza de una ma-

nera transparente al programador que por lo tanto se ahorra el tener que escribir una gran cantidad de líneas de control.

1.2 Introducción a Window:

El AWT proporciona dos tipos de contenedores, ambos son implementados como subclases de Container (que es una subclase de Component). Las subclases de Windows : Dialog, FileDialog, y Frame proporcionan ventanas para contener componentes. La clase Frame crea ventanas normales y completamente maduras, como oposición a las creadas por la clase Dialogs, que son dependientes del marco y pueden ser modales. Los Paneles agrupan los componentes dentro de un área de una ventana existente.

2. Concepto de Layouts Managers:

El AWT define cinco Layout Managers: dos muy sencillos (FlowLayout y GridLayout), dos más especializados (BorderLayout y CardLayout) y uno muy general GridBagLayout). Además, los usuarios pueden escribir su propio Layout Manager, implementando la interface LayoutManager, que especifica 5 métodos.

Los primeros tres despliegues tienen dos virtudes importantes: la sencillez y la automatización. El último tiene como principal virtud la flexibilidad. Todos están diseñados para realizar cierta presentación automáticamente y sin necesidad de colocar cada componente exactamente en una posición numérica o manualmente, como suele hacerse en otros lenguajes o en entornos de programación visual.

Java permite también posicionar los Components de modo absoluto, sin Layout Manager, pero de ordinario puede perderse la portabilidad y algunas otras características. Todos los Containers tienen un Layout Manager por defecto, que se utiliza si no se dice otra cosa: Para Panel, el defecto es un objeto de la clase FlowLayout. Para Window, Frame y Dialog, el defecto es un objeto de la clase BorderLayout.

2.1 Ventajas y Desventajas de los Layout Managers:

Los layout managers tienen una gran cantidad de ventajas:

- Encapsulan parte de la lógica de presentación de nuestro interfaz gráfico de modo que evitan al programador tener que escribir una gran cantidad de líneas de código. Además hacen este código mucho más sencillo de leer y por lo tanto más mantenible.

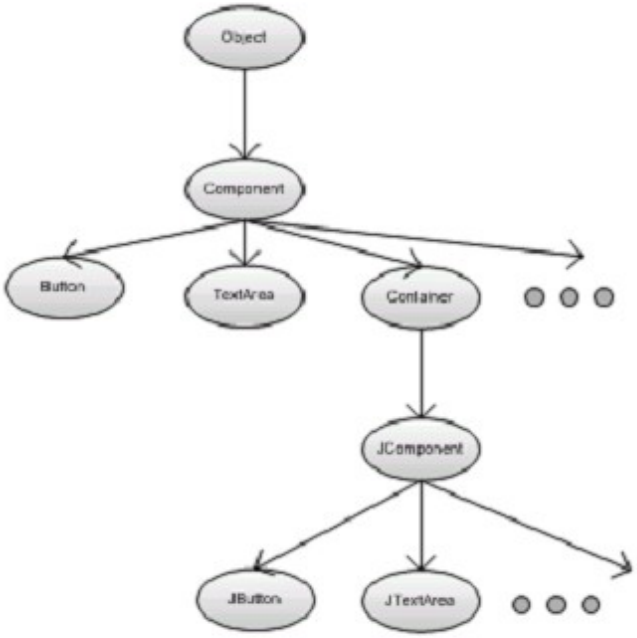
- Reorganizan automáticamente los componentes del interfaz de modo que siempre se ajuste a las directivas que hemos establecido previamente. Si el usuario en un momento dado decide maximizar el interfaz gráfico éste mantendrá su aspecto original en la medida de lo posible. De este modo no limitamos al usuario a un formato de pantalla determinado.
- Hacen más sencillo la labor de añadir, modificar y eliminar componentes. En un diseño tradicional cuando nos vemos obligados a añadir un componente en un lugar donde ya existen varios, seguramente tengamos que mover el resto de componentes del interfaz gráfico para acomodar a nuestro nuevo inquilino. Utilizando layouts managers, lo único que tenemos que hacer es agregar el componente y el se encarga automáticamente de reorganizar todo el interfaz.
- Hacen nuestro interfaz mucho más portable. Esto se debe a que los componentes gráficos no tienen las mismas propiedades en todos los sistemas operativos. Un botón que muestre la cadena "Hola Mundo" en Mac no tendrá las mismas dimensiones que su homónimo en Linux o Windows. Al realizar nuestro programa con layouts managers , éstos ya se encargan de ajustar los componentes adecuadamente y nos evitamos problemas inesperados.
- Ciertamente existen también una serie de desventajas asociadas a los layout managers
- Requieren una buena dosis de aprendizaje y práctica. Muchos programadores odiarán los layouts managers ya que pueden resultar una experiencia traumática sin un buen aprendizaje. De todos modos, una vez dominados, son pocos los programadores que dejan de utilizarlos.
- Pueden generar interfaces muy pesadas. A menudo las primeras veces que creemos nuestros layouts nos encontraremos con que acabamos con un gran número de paneles anidados. Los paneles son objetos bastante pesados por lo que hay que tener cuidado de no sobrecargar innecesariamente nuestra interfaz gráfica.

2.2 Contenedores y Componentes

Para poder entender el funcionamiento de los layout manager es necesario una pequeña base sobre lo que son los contenedores y los componentes.

Un **contenedor** es un componente Java que puede contener otros componentes. La clase principal es java.awt.Component de la cual se heredan componentes como java.awt.Button, java.awt.Label, etc..., y también se hereda la clase java.awt.Container que representa a un objeto contenedor.

En la siguiente figura podemos ver un extracto de la jerarquía de clases de AWT y swing.



Los layouts por defecto para los contenedores de swing son los mismos que para sus homónimos de AWT. Si queremos cambiar el layout manager de un contenedor en un momento dado, tan sólo tendremos que llamar al método:

```
contenedor.setLayout(LayoutManager layout);
```

Si en cualquier momento decidimos que estamos hartos de un layout manager y queremos encargarnos nosotros mismos de la gestión de componentes tan sólo tendremos que escribir:

```
contenedor.setLayout(null);
```

Los componentes como hemos dicho son objetos que forman parte de nuestro interfaz gráfico. Cada componente tiene asignada una coordenada horizontal, una coordenada vertical, una longitud y una anchura determinadas. Estos valores serán los que se utilizarán para renderizar el componente en pantalla.

La clase java.awt.Component nos ofrece una serie de métodos para poder modificar los valores de los atributos anteriores:

```
public void setSize(Dimension size);
```

```
public void setBounds(Rectangle r);
```

Hay pequeñas variaciones de estos métodos pero su función es la misma. Por su parte la clase javax.swing.JComponent tiene métodos diferentes:

```
public void setPreferredSize(Dimension size);
```

```
public void setMinimumSize(Dimenstion size);
```

```
public void setMaximumSize(Dimension size);
```

Debemos tener en cuenta que cuando utilizamos layout managers, no sirve de nada intentar establecer a la fuerza el tamaño de los componentes ya que será el layout el que siempre tenga la última palabra. Técnicamente existen formas de forzar un tamaño a los componentes aunque se estén utilizando layouts, pero no tiene ningún sentido el hacerlo ya que perderámos la ventaja que éstos nos ofrecen.

El último aspecto importante a tener en cuenta es el significado del atributo preferredSize. Este atributo indica el tamaño que a un componente le gustaría tener. Este tamaño suele ser dependiente de la plataforma y el layout manager intentará respetarlo siempre que sea posible y siempre que lo permitan sus políticas de layout.

2.3 Reglas Generales para el uso de Controladores de Disposición

Se debe elegir el Layout Manager que mejor se adecúe a las necesidades de la aplicación que se desea desarrollar. Recuérdese que cada Container tiene un Layout Manager

por defecto. Si se desea utilizar el Layout Manager por defecto basta crear el Container (su constructor crea un objeto del Layout Manager por defecto e inicializa el Container para hacer uso de él).

Para utilizar un Layout Manager diferente hay que crear un objeto de dicho Layout anager y pasárselo al constructor del container o decirle a dicho container que lo utilice por medio del método setLayout().

- **Cómo Utilizar FlowLayout:**

FlowLayout es el controlador por defecto para todos los Paneles. Simplemente coloca los componentes de izquierda a derecha, empezando una nueva línea si es necesario.

- **Cómo Utilizar BorderLayout:**

BorderLayout es el controlador de disposición por defecto para todas las ventanas, como Frames y Cuadros de Diálogo. Utiliza cinco áreas para contener los componentes: north, south, east, west, and center (norte, sur, este, oeste y centro). Todo el espacio extra se sitúa en el área central.

- **Cómo Utilizar CardLayout:**

Utiliza la clase CardLayout cuando tengas un área que pueda contener diferentes componentes en distintos momentos. Los CardLayout normalmente son controlados por un objeto Choice, con el estado del objeto, se determina que Panel (grupo de componentes) mostrará el CardLayout.

- **Cómo utilizar GridLayout:**

GridLayout simplemente genera una matriz de Componentes que tienen el mismo tamaño, mostrándolos en una sucesión de filas y columnas.

- **Cómo Utilizar GridBagLayout:**

GridBagLayout es el más sofisticado y flexible controlador de disposición proporcionado por el AWT. Alínea los componentes situándolos en una parrilla de celdas, permitiendo que algunos componentes ocupen más de una celda. Las filas de la parrilla no tienen porque ser de la misma altura; de la misma forma las columnas pueden tener diferentes anchuras.

3.Flow Layout

El primer que vamos a ver es también el más simple. FlowLayout coloca componentes en filas horizontales. FlowLayout es el layout manager por defecto de paneles y applets.

FlowLayout respeta siempre el tamaño preferido de cada componente. Cuando queremos insertar un componente y no hay más espacio en la fila actual, el elemento se insertará en la fila siguiente. Los componentes de cada fila se encuentran equiespaciados. Pode-

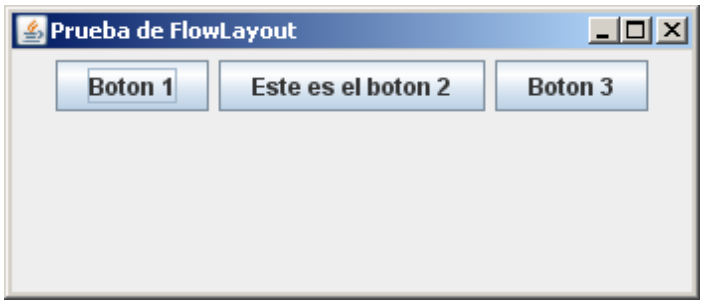
mos controlar la alineación de los elementos en las filas utilizando los atributos estáticos FlowLayout.LEFT, FlowLayout.CENTER y FlowLayout.RIGHT.

Por defecto FlowLayout deja un espacio de cinco puntos tanto horizontal como vertical entre componentes. FlowLayout tiene varios constructores con los que podemos modificar este espaciado y también la alineación de los componentes.

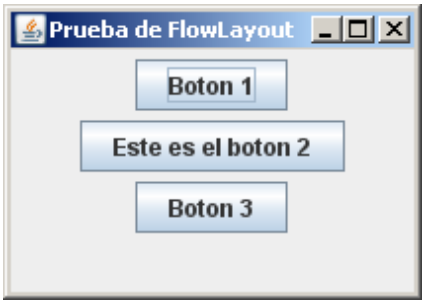
Veamos un ejemplo de funcionamiento de este layout manager:

```
import javax.swing.*;
public class Ejemplo_FlowLayout extends JFrame{
public static void main(String args[]){
   Ejemplo_FlowLayout frame = new Ejemplo_FlowLayout();
   JPanel panel = new JPanel();
   JButton boton1 = new JButton("Boton 1");
   JButton boton2 = new JButton("Este es el boton 2");
   JButton boton3 = new JButton("Boton 3");
   panel.add(boton1);
   panel.add(boton2);
   panel.add(boton3);
   frame.setContentPane(panel);
   frame.setSize(350,150);
   frame.setTitle("Prueba de FlowLayout");
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   frame.setVisible(true);
} }
```

Al compilar y ejecutar este programa nos debería salir una ventana como la de la figura.



Si modificamos las dimensiones de la ventana podemos ver como los elementos ser re-organizan según el espacio disponible.



4.BorderLayout

BorderLayout es el layout manager por defecto para frames por lo que al igual que FlowLayout su aprendizaje es indispensable. BorderLayout divide el espacio de un contenedor en cinco regiones diferentes. Estas regiones son: North, South, East, West y Center, y se corresponden con su situación dentro del contenedor en el que se encuentran.

Veamos más claramente lo que quiere decir esto con un ejemplo sencillo:


```
import javax.swing.*;
import java.awt.*;
public class Ejemplo_BorderLayout extends JFrame{
```

IES de Teis
Avda. de Galicia, 101
36216 – Vigo
Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es


```
    public static void main(String[] args){
        Ejemplo_BorderLayout frame = new Ejemplo_BorderLayout();
        Container Mipanel = frame.getContentPane();
        JButton norte = new JButton("Norte");
        JButton sur = new JButton("Sur");
        JButton este = new JButton("Este");
        JButton oeste = new JButton("Oeste");
        JButton centro = new JButton("Centro");
        Mipanel.add(norte,BorderLayout.NORTH);
        Mipanel.add(sur,BorderLayout.SOUTH);
        Mipanel.add(este,BorderLayout.EAST);
        Mipanel.add(oeste,BorderLayout.WEST);
        Mipanel.add(centro,BorderLayout.CENTER);
        frame.setSize(350,150);
        frame.setTitle("Prueba de BorderLayout");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
    }
```

Como podemos ver en el código fuente anterior, al añadir un elemento al BorderLayout tenemos que especificar la región en la cual lo queremos añadir. Si no especificamos ninguna región por defecto el componente se inserta en el centro del contenedor.

Si insertamos un componente en una región donde había otro componente previamente, el que se encontraba en el contenedor desaparecerá y el nuevo ocupará su lugar. Por lo tanto tenemos que tener cuidado con donde insertamos los componentes.

Para finalizar con este layout manager vamos a hablar de como trata el tamaño de los componentes. Cuando añadimos un componente en las posiciones norte o sur, BorderLayout respeta su alto mientras que la longitud del mismo se ajusta hasta ocupar todo el ancho del contenedor. Con los componentes de las posiciones este y oeste pasa lo contrario. Por último el objeto que se situe en la zona central ocupará el resto de espacio disponible.

IES de Teis
Avda. de Galicia, 101
36216 – Vigo
Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es

5. CardLayout

CardLayout es una layout manager ligeramente diferente a todos los demás ya que tan sólo muestra en un instante dado un único componente. Un contenedor que tenga asignado un CardLayout podrá tener cualquier número de componentes en su interior pero sólo uno se verá en un instante dado.

En este layout manager los componentes ocuparán todo el tamaño disponible en el contenedor. Los componentes a medida que se insertan en el contenedor van formando una secuencia. Para seleccionar el componente que queremos mostrar en cada momento disponemos de varios métodos:

```
public void first(Container contenedor);
public void last(Container contenedor);
public void next(Container contenedor);
public void previous(Container contenedor);
public void show(Container contenedor, String nombre);
```

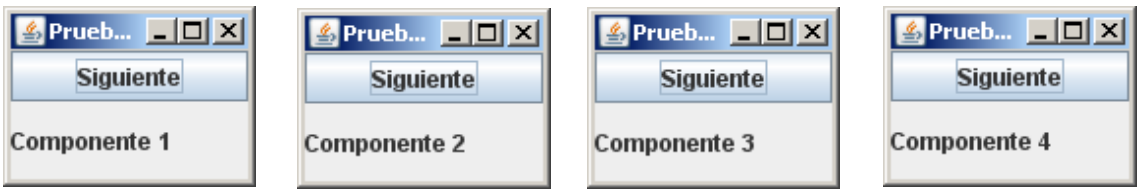
Al añadir componentes tendremos que fijarnos en que el orden en el que los añadamos al contenedor será el orden en el que serán recorridos por el layout manager.

Veamos un ejemplo simple:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Ejemplo_CardLayout extends JFrame{
   public static void main(String[] args){
      Ejemplo_CardLayout frame = new Ejemplo_CardLayout();
      Container miContenedor = frame.getContentPane();
      JButton siguiente = new JButton("Siguiente");
      miContenedor.add(siguiente, BorderLayout.NORTH);
      JLabel label1= new JLabel("Componente 1");
      JLabel label2= new JLabel("Componente 2");
      JLabel label3= new JLabel("Componente 3");
      JLabel label4= new JLabel("Componente 4");
```

```
        final JPanel miPanel= new JPanel();
        final CardLayout layout=new CardLayout();
        miPanel.setLayout(layout);
        miPanel.add(label1,"1");
        miPanel.add(label2,"2");
        miPanel.add(label3,"3");
        miPanel.add(label4,"4");
        miContenedor.add(miPanel,BorderLayout.CENTER);
        siguiente.addActionListener(new ActionListener(){
           public void actionPerformed(ActionEvent e){
              layout.next(miPanel);
           }       });
        frame.setSize(350,150);
        frame.setTitle("Prueba de CardLayout");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

El ejemplo es muy sencillo y muestra un contenedor con varias etiquetas. Con un botón podemos ir avanzando de etiqueta. Al pulsar el botón se muestra la etiqueta siguiente llamando al método next() de CardLayout.



Este layout manager es muy sencillo y muy útil especialmente cuando tenemos un panel que variará su contenido en función de alguna parte de nuestro interfaz. En lugar de eliminar el panel e insertar otro nuevo, o en lugar de eliminar los componentes e insertar otros nuevos, podemos utilizar un CardLayout que nos ahorra gran cantidad de trabajo.

IES de Teis
Avda. de Galicia, 101
36216 – Vigo
Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es

6. GridLayout:

GridLayout divide el espacio de un contenedor en forma de tabla, es decir, en un conjunto de filas y columnas. Cada fila y cada columna tiene el mismo tamaño y el área del contenedor se distribuye equitativamente entre todas las celdas. De todo esto se deduce que GridLayout no respetará el tamaño preferido de los componentes que insertemos en cada una de las celdas.

El número de filas y columnas se especifica en el constructor. Si pasamos cero como el número de filas o columnas, el layout manager irá creando las filas y columnas en función del número de componentes y del valor de la otra dimensión, es decir, si creamos un GridLayout con cero filas y tres columnas e insertamos cuatro componentes el GridLayout será lo suficientemente inteligente como para saber que tiene que crear dos filas.

Veamos un ejemplo de funcionamiento:

```
import javax.swing.*;
import java.awt.*;

public class Ejemplo_GridLayout extends JFrame{
   public static void main(String[] args){
      Ejemplo_GridLayout frame = new Ejemplo_GridLayout();
      Container miContenedor = frame.getContentPane();
      int X =3; int Y= 3;
      miContenedor.setLayout(new GridLayout(X,Y));
      for (int i=0;i<X;i++){
         for (int j=0; j<Y;j++)
            miContenedor.add(new JButton(i+" x "+j));
      }
      frame.setSize(400,300);
      frame.setTitle("Prueba de GridLayout");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
}
```

```
}
```

En la figura anterior vemos el resultado de ejecutar el ejemplo. Como se puede ver todas las celdas tienen el mismo tamaño y los botones ocupan la totalidad de la celda.

GridLayout es un layout manager realmente muy simple y como tal su utilidad se encuentra bastante reducida. Suele ser útil a la hora de crear partes del interfaz de usuario que necesiten representar una matriz de componentes o incluso para interfaces que tengan en si una forma matricial.

7. GridBagLayout:

GridBagLayout es el layout manager más poderoso y eficaz con mucha diferencia. Con GridBagLayout podemos imitar facilmente el comportamiento del resto de layout mangers a parte de poder crear con él interfaces mucho más complejas.

▪Ventajas y desventajas:

GridBagLayout es el layout manager que más pavor causa entre los programadores Java. Odiado por unos y alabado por otros, este layout manager proporciona una serie de ventajas sobre el resto:

- Permite la creación de interfaces de usuario complejos. Con este layout manager tenemos control absoluto sobre las posiciones que ocuparán los objetos en el interfaz final.

- Las interfaces construidas son más ligeras. Cuando queremos crear un interfaz de usuario combinando el resto de layout managers vistos hasta el momento a menudo terminamos con un número grande de paneles anidados. Los paneles son objetos bastante pesados y tener gran cantidad de los mismos puede influir perjudicialmente en el rendimiento de nuestro programa. Con GridBagLayout se pueden crear interfaces exactamente iguales pero con un único panel con lo que nuestra interfaz será mucho más ligera.

Pero como todo, también tiene sus inconvenientes:

- Requiere un tiempo de aprendizaje bastante grande. No sólo es necesario comprender su funcionamiento sino que también es necesario haber hecho bastantes ejemplos para llegar a dominarlo.
- El código necesario para crear una interfaz de usuario es considerablemente más grande que con el resto de layout managers y además suele ser un código bastante complicado y difícil de comprender y por tanto de mantener.

7.1 GridBagLayout a fondo:

GridBagLayout basa su funcionamiento en una clase auxiliar que establece restricciones a los componentes, GridBagConstraints. Estas restricciones especifican exactamente como se mostrará cada elemento dentro del contenedor. La clase GridBagConstraints posee bastantes atributos que nos permiten configurar el layout de un contenedor a nuestro gusto.

A continuación vamos a ver los atributos más importantes:

- **gridx y gridy:**

Estos dos atributos especifican las coordenadas horizontal y vertical del componente que vamos a insertar en el grid. Realmente no siempre es necesario establecer su valor ya que en los casos más simples nos llegaría con **gridwidth** y **gridheight,** sin embargo la experiencia dice que poner este atributo es recomendable ya que permite saber en que elemento nos encontramos de una manera visual.

- **gridwidth y gridheight:**

Este otro par de elementos junto con **gridx** y **gridy** son la base de GridBagLayout. Comprendiendo a la perfección su significado no habrá ningún interfaz que se nos resista. Básicamente lo que indican es el número de celdas que ocupará un componente dentro del GridBagLayout. Su valor puede ser:

- Un número cardinal, en este caso indica exactamente el número de filas o columnas que ocupará el componente.
- GridBagConstraints.RELATIVE, indica que el componente ocupará el espacio disponible desde la fila o columna actual hasta la última fila o columna disponible.
- GridBagConstraints.REMAINDER, indica que el componente es el último de la fila actual o columna actual.

Como podemos ver, gridwidth y gridheight especifican el número de celdas horizontal y vertical que abarcará un componente. Además podemos utilizar los valores especiales RELATIVE y REMAINDER para indicar que un componente ha de ocupar todo el espacio restante o todo el espacio hasta el último componente.

anchor:

Este atributo es mucho más sencillo; anchor especifica la posición que ocupará un componente dentro de una celda. Los valores que puede tomar este atributo están definidos como variables estáticas dentro de la clase GridBagConstraints y son: NORTH, SOUTH, EAST, WEAST, NORTHWEST, SOUTHWEST, NORTHEAST, SOUTHEAST y CENTER. Indican la orientación de los componentes dentro de la celda que ocupan.

fill:

El atributo fill especifica el espacio que ocupará el componente dentro de la celda. Los valores que puede tomar son variables estáticas de la clase GridBagConstraints.

- NONE: El componente ocupará exactamente el tamaño que tenga como preferido.
- HORIZONTAL: El componente ocupará todo el espacio horizontal de la celda mientras que su altura será la que tenga como preferida.
- VERTICAL: El componente ocupará todo el espacio vertical de la celda mientras que su longitud será la que tenga como preferida.
- BOTH: El componente ocupará la totalidad de la celda.

weightx y weighty:

A mediada que vamos añadiendo componentes a un contenedor el layout manager va determinando en función del tamaño de los componentes el espacio que ocupan las celdas. Hay que tener mucho cuidado porque al contrario de lo que pueda parecer si no indicamos nada las celdas no ocuparán la totalidad del contenedor. ¿Cómo hacemos para que las celdas ocupen la totalidad del contenedor?.

Los atributos weightx y weighty especifican el porcentaje de espacio libre que ocupará una celda determinada.

El espacio libre se dividirá entre todas las celdas que especifiquen valores dentro de los atributos weightx y weighty. La forma de especificar el espacio que quiere ocupar cada

componente es mediante un número entre 0.0 y 1.0. Este número representa al porcentaje de espacio libre que ocupará cada celda.

Un ejemplo:

- Espacio libre 250 puntos en horizontal y 150 puntos en vertical.
- Componente 1: c.weightx=1.0, c.weighty=1.0
- Componente 2: c.weightx=0.4, c.weighty=1.0

Como ambos componentes han pedido espacio libre, éste se divide entre ambos, por lo tanto a cada uno le tocan 125 puntos. Sin embargo como el componente 2 tan sólo quiere el 40% del espacio libre se le asignan 50 puntos y el resto de los puntos pasan al otro componente que recibirá sus 125 puntos más los 75 puntos que sobraron del segundo componente, en total 200 puntos.

El espacio vertical es más sencillo. Como vimos antes se divide el espacio libre entre los componentes que lo han pedido. En este caso como los dos componentes han pedido la totalidad de su parte, a ambos les corresponden 75 puntos.

Obviamente cuando estemos diseñando el interfaz no estaremos pensando en si este componente va a tener unos puntos y otro otros, sin embargo los atributos weightx y weighty son de una ayuda estimable para hacer que determinadas partes de nuestra interfaz sean más grandes que las otras.

- **insets:**

El atributo insets es un objeto de la clase java.awt.Insets cuyo constructor es:

```
Insets(int top, int left, int bottom, int right);
```

Los parámetros del constructor especifican el espacio que se dejará de margen. Según el valor de insets con que insertemos el componente se pondrá uno u otro margen.

7.2 Ejemplo GridBagLayout :

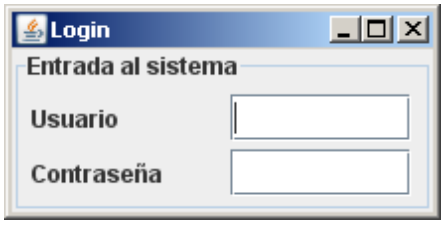
Vamos a implementar el siguiente ejemplo sencillo, de una ventana de entrada al sistema. Veamos como sería el código para crear esta interfaz.

```
import javax.swing.*;

import java.awt.*;
```

```
public class FormularioInicio {

    public static void main (String[] args){

        JFrame f= new JFrame();

        Container contenedor= f.getContentPane();

        contenedor.setLayout(new GridBagLayout());

        ((JPanel)contenedor).setBorder(BorderFactory.createTitledBorder("Entrada al
                sistema"));

        GridBagConstraints c= new GridBagConstraints();

        c.weightx=0.4; c.weighty=1.0;

        c.gridwidth=GridBagConstraints.RELATIVE;

        c.gridheight=GridBagConstraints.RELATIVE;

        c.fill=GridBagConstraints.BOTH;

        c.anchor=GridBagConstraints.WEST;

        c.insets=new Insets(2,5,2,0);

        contenedor.add(new JLabel("Usuario"),c);

        c.gridwidth=GridBagConstraints.REMAINDER;

        c.gridheight=GridBagConstraints.RELATIVE;

        c.weightx=1.0;

        c.insets=new Insets(2,0,2,5);

        contenedor.add(new JTextField(),c);

        c.gridwidth=GridBagConstraints.RELATIVE;

        c.gridheight=GridBagConstraints.REMAINDER;

        c.weightx=0.4;

        c.insets=new Insets(2,5,2,0);

        contenedor.add(new JLabel("Contraseña"),c);

        c.gridwidth=GridBagConstraints.REMAINDER;

        c.gridheight=GridBagConstraints.REMAINDER;

        c.weightx=1.0;

        c.insets=new Insets(2,0,2,5);

        contenedor.add(new JTextField(),c);
```

```
        f.setSize(220,110);
        f.setTitle("Login");
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Como se puede ver cada vez que añadimos un componente al contenedor hemos de pasar una variable de tipo GridBagConstraints al método add. No se debe olvidar pasar la variable de constraints porque en otro caso el interfaz no tendrá el aspecto esperado. La ventana será la siguiente.



8. Ventanas, Marcos y Cuadros de Diálogo:

La programación gráfica de aplicaciones Java que no son applets, requiere de una clase especial llamada **Window** y de dos extensiones suyas que se usan con mayor frecuencia que **Window: Frame** y **Dialog.** Los objetos de la clase **Window** se llaman ventanas.

La ventana no tiene orilla, ni título, ni botón de salida, el cuadro de diálogo sí tiene orilla, título y botón de salida, pero no tiene botones de minimizar y maximizar ni un menú. Finalmente el marco tiene todo: orilla, título, botones de minimizar y maximizar y una barra de menú. Estas son precisamente las características que más distinguen entre sí a las ventanas, los cuadros de diálogo y los marcos.

El marco ofrece más funcionalidad que el cuadro de diálogo y éste más que una ventana. Por este motivo los marcos se utilizan con mayor frecuencia y las ventanas puras sólo de vez en cuando.

8.1 Clase Window:

Los objetos de la clase Windows son ventanas de máximo nivel, pero sin bordes y sin barra de menús. En realidad son más interesantes las clases que derivan de ellas: Frame y Dialog.

Métodos de Window	Función que realizan
toFront(), toBack()	Para desplazar la ventana hacia delante y hacia atrás en la pantalla.
setVisible(boolean)	Muestra u oculta la ventana
pack()	Hace que los componentes se reajusten al tamaño preferido.

8.2 Clase Frame:

La clase Frame es una ventana con un borde y que puede tener una barra de menús. Si una ventana depende de otra ventana, es mejor utilizar una Window que un frame. La siguiente tabla muestra algunos métodos más utilizados de la clase Frame.

Además de los métodos citados, se utilizan mucho los métodos setVisible(boolean), pack(), toFront() y toBack(), heredados de la superclase Window.

Métodos de Frame	Función que realizan
Frame(), Frame(String title)	Constructores de frame.
String getTitle(), setTitle (String)	Obtienen o determinan el título de la ventana.
MenuBar getMenuBar(), setMenuBar(MenuBar).	Permite obtener, establecer la barra de menús.
Image getIconImage(), setIconImage(Image)	Obtienen o determinan el icono que aparecerá en la barra de títulos.
setResizable(boolean), boolean isResizable()	Determinan o chequean si se puede cambiar el tamaño.

8.3Clase Dialog:

Un Dialog es una ventana que depende de otra ventana (de una Frame). Lo único que distingue a los cuadros de diálogo de las ventanas normales (que son implementadas con objetos Frame) es que el cuadro de diálogo depende de alguna otra ventana (un Frame). Cuando esta otra ventana es destruida, también lo son sus cuadros de diálogo dependientes. Cuando esta otra ventana es miniaturizada sus cuadros de diálogo desaparecen de la pantalla. Cuando esta otra ventana vuelve a su estado normal, sus cuadros de diálogo vuelven a aparecer en la pantalla. El AWT proporciona automáticamente este comportamiento. Como no existe un API actualmente que permita a los Applets encontrar la ventana en la que se están ejecutando, estos generalmente no pueden utilizar cuadros de diálogo. La excepción son los applets que traen sus propias ventanas (Frames) que pueden tener cuadros de diálogo dependientes de esas ventanas.

Los cuadros de diálogo pueden ser modales. Los cuadros de diálogo modales requieren la atención del usuario, para evitar que el usuario haga nada en la aplicación del cuadro de diálogo hasta que se haya finalizado con él. Por defecto, los cuadros de diálogo no son modales, el usuario puede mantenerlos y seguir trabajando con otras ventanas de la aplicación.

La siguiente tabla muesta los métodos más importantes de la clase Dialog. Se pueden utilizar también los métodos de sus superclases.

Métodos de Dialog	Función que realizan
Dialog(Frame fr), Dialog(Frame fr, boolean mod), Dialog(Frame fr,String title) Dialog(Frame parent,String title, boolean mod)	Constructores.
int getTitle(), void setTitle(String)	Permite obtener o determinar el título.
boolean isModal (), setModal(boolean)	Pregunta o determina si el Dialog es modal o no.
boolean isResizable(), setResizable(boolean)	Pregunta o determina si puede cambiar el tamaño.

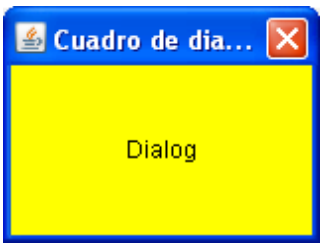
8.4Clase FileDialog

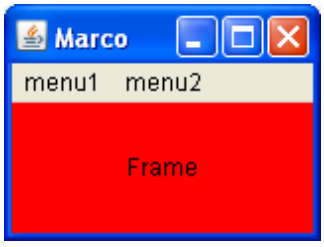
La clase FileDialog muestra una ventana de diálogo en la cual se puede seleccionar un fichero. Esta clase deriva de Dialog.

Métodos de FileDialog	Función que realizan
FileDialog(Dialog parent), FileDialog(Dialog parent,String title) FileDialog(Dialog parent,String title, int mode) FileDialog(Frame parent) FileDialog(Frame parent,String title) FileDialog(Frame parent,String title, int mode)	Constructores.
int getMode(), void setMode(int mode)	Modo de apertura (SAVE o LOAD)
String getDirectory(), String getFile()	Obtiene directorio o fichero elegido.
void setDirectory(String dir), void setFile(String file)	Determina el directorio o fichero elegido.
FilenameFilter getFilenameFilter(), void setFilenameFilter(FilenameFilter filter)	Determina o establece el filtro para los ficheros.

8.5 Ejemplo práctico:

A continuación, vamos a través de un ejemplo, comprobar las diferencias entre cada una de las Windows; ventana, marco o frame y cuadro de diálogo.





Creación de una ventana

```
import java.applet.*;
```

```
import java.awt.*;
public class Ventana extends Window {
    Ventana(Frame fr,int x,int y,int dx, int dy) {
            super(fr);
            Label lbl=new Label("Window",Label.CENTER);
            lbl.setBackground(Color.cyan);
            add("Center",lbl);
            pack();
            setSize(dx,dy);
            setLocation(x,y);
            show();
    }
}
```

Creación de un marco

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class Marco extends Frame{
    Marco(String título,int x,int y,int dx,int dy) {
            super(título);
            setBackground(Color.red);
            Label lbl=new Label("Frame",Label.CENTER);
            setBackground(Color.red);
            add("Center",lbl);
            MenuBar menubar=new MenuBar();
            setMenuBar(menubar);
            Menu menu1=new Menu("menu1");
            menu1.add(new MenuItem("item 1.1"));
            menu1.add(new MenuItem("item 1.2"));
            menu1.add(new MenuItem("item 1.3"));
```

```
            Menu menu2=new Menu("menu2");
            menu2.add(new MenuItem("item 2.1"));
            menu2.add(new MenuItem("item 2.2"));
            menubar.add(menu1);
            menubar.add(menu2);
            pack();
            setSize(dx,dy);
            setLocation(x,y);
            setVisible(true);
    }
```

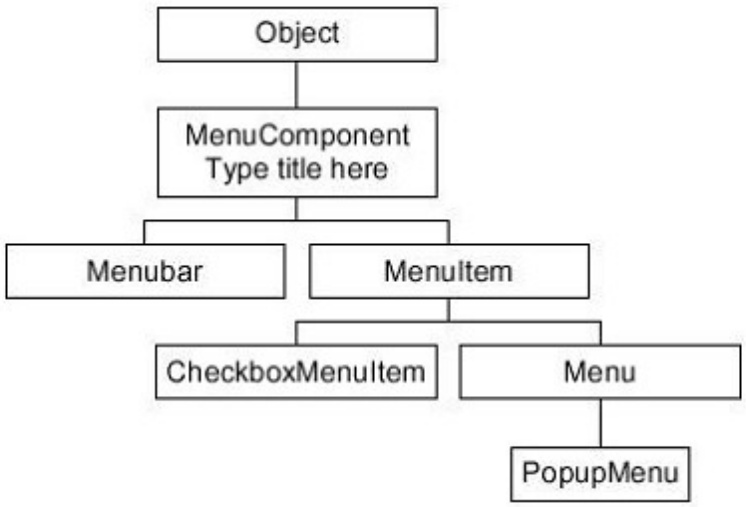
Creación de un cuadro de diálogo

```
import java.applet.*;
import java.awt.*;
public class Dialogo extends Dialog {
    Dialogo(Frame madre,String título,boolean modal,int x,int y,int dx,int dy) {
            super(madre,título,modal);
            Label lbl=new Label("Dialog",Label.CENTER);
            lbl.setBackground(Color.yellow);
            add("Center",lbl);
            pack();
            setSize(dx,dy);
            setLocation(x,y);
            setVisible(true);
    }
}
```

9.Menús:

Los Menús de Java no descienden de Component, sino de MenuComponent, pero tienen un comportamiento similar, pues trabajan con eventos. La siguiente figura muestra la

jerarquía de clases de los menús de JAVA.



Para crear un menú se debe crear primero un MenuBar; después se crean los menús de MenuItem. Los MenuItems se añaden al menu correspondiente; los menús se añaden al MenuBar y el MenuBar se añade a un Frame.

Cada menú está representado por un objeto Menu. Esta clase está implementada como una subclase de MenuItem para se puedan crear submenús fácilmente añadiendo un menú a otro MenuBar

Las barras de menú están implementadas por la clase MenuBar. Esta clase representa una noción dependiente de la plataforma de un grupo de manús adheridos a una ventana. Las barras de menú no no pueden utilizarse con la clase Panel.

9.1 Construcción de un menú.

Las **barras de menús** sólo pueden aparecer en un marco (Frame) y cada marco sólo puede tener una barra de menús. Para agregar una barra de menús a un marco basta crearla y asignarla al marco mediante la función setMenuBar. Es decir, basta escribir estas dos líneas:

```
MenuBar mb=new MenuBar();
setMenuBar(mb);
```

dentro de la construcción del marco. El método setMenuBar es un método de la clase Frame. A una *barra de menús* se le pueden agregar **menús**. Para ello hay que construir cada menú dándole un nombre.

IES de Teis
Avda. de Galicia, 101
36216 – Vigo
Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es

Por ejemplo:

```
Menu archivo=new Menu("archivo");
Menu edición=new Menu("edición");
```

crea dos menús de nombre "archivo" y "edición" respectivamente. Estos dos menús pueden agregarse a la barra de menús mb de esta manera:

```
mb.add(archivo);
mb.add(edición);
```

Finalmente, a los menús se les pueden agregar *elementos de menú* (MenuItem) que se crean también asignándoles un nombre a cada uno. Por ejemplo:

```
MenuItem nuevo=new MenuItem("nuevo");
MenuItem abrir=new MenuItem("abrir");
MenuItem guardar=new MenuItem("guardar");
```

crea tres elementos de menú, los cuales pueden agregarse, por ejemplo, a archivo de esta manera:

```
archivo.add(nuevo);
archivo.add(abrir);
archivo.add(guardar);
```

Para crear una línea de separación dentro de un menú se agrega un elemento de menú de nombre "-" (signo menos) así;

```
archivo.add(new MenuItem("-"));
```

o simplemente:

```
archivo.add("-");
```

pues el método Menu.add(String cadena) crea un elemento de menú con nombre cadena y lo agrega al menú.

La instrucción

```
nuevo.setEnabled(false);
```

haría que la opción "nuevo" apareciera deshabilitada.

A un menú se le puede agregar otro menú así:

```
opciones=new Menu(opciones);
edición.add(opciones);
```