# Conocimiento: Gestor de Diseño

**Definición**

Para ubicar los componentes dentro del contenedor, java provee un conjunto de clases que están en la categoría de Gestores de Diseño o Layout Manager.

Estas clases se encargan de:

* Definir tamaño del contendor
* Definir ubicación y distribución de cada componente
* Manejar las operaciones de agregar, modificar, mover y eliminar componentes.

Todos los contenedores de AWT tienen asociado un gestor de diseño o layout manager por defecto, según el siguiente cuadro:

|  |  |
| --- | --- |
| **Contenedor** | **Layout manager** |
| Panel | FlowLayout |
| Applet | FlowLayout |
| Frame | BorderLayout |
| Dialog | BorderLayout |
| ScrollPane | FlowLayout |

Los layout por defecto para los contenedores de swing son los mismos que para sus homónimos de AWT.

Cada layout se caracteriza por el estilo que emplea para situar los componentes en su interior, es decir:

* Alineación de izquierda a derecha
* Alineación en rejilla
* Alienación del frente a atrás.

Los gestores de diseño permiten que la aplicación sea independiente de la resolución de los sistemas donde se ejecute.

Para cambiar el gestor de diseño los componentes tiene definido el método setLayout(Layout manager), que permite cambiar el gestor de diseño del contenedor por defecto.

En AWT existen los siguientes gestores de diseño:

1. FlowLayout
2. BorderLayout
3. CardLayout
4. GridLayout
5. GridBagLayout

**FlowLayout**

Esta clase pertenece al paquete java.awt y permite organizar los componentes dentro del contenedor de izquierda a derecha. Esta clase modifica la posición del componente pero no modifica el tamaño.

Además se puede definir la alineación de los componentes, que por defecto es centrada.

Ejemplo de uso de FlowLayout en un Frame, con alineación por defecto:



El resultado de la ejecución del código es:



Cada botón se alinea desde el centro hacia la izquierda y hacia la derecha.

Qué pasaría si le decimos que los componentes se alineen a la izquierda:



Ahora, el resultado de la ejecución del código es:



Ahora los botones se alinean desde la izquierda.

**BorderLayout**

Esta clase pertenece al paquete java.awt y permite organizar los componentes en un esquema basado en los puntos cardinales. Permite modificar posición de cada componente pero no los tamaños.

El siguiente esquema muestra la organización que utiliza el BorderLayout y la constante de la clase que se hace cargo de definir cada posición dentro de este gestor:



BorderLayout.NORTH

BorderLayout.EAST

BorderLayout.WEST

BorderLayout.CENTER

BorderLayout.SOUTH

Ejemplo de uso de BorderLayout en un Frame:



El resultado de la ejecución del código es:



Aquí se observa que cada botón queda en un área distinta del contenedor.

**CardLayout**

Esta clase pertenece al paquete java.awt y permite implementar un área que contiene diferentes componentes en diferentes ocasiones.

Ejemplo de uso de CardLayout en un frame:



En este código se define un objeto CardLayout, luego se agregan componentes al Frame identificándolos con un label, luego se define cuál componente será desplegado por el gestor de diseño.

El resultado de la ejecución del código es:



 En la ventana se aprecia el boton2 que fue el escogido para ser mostrado.

**GridLayout**

Esta clase pertenece al paquete java.awt y permite distribuir componentes en celdas de igual tamaño, formando filas y columnas.

Si el Contenedor es alterado en su tamaño en tiempo de ejecución, el sistema intentará mantener el mismo número de filas y columnas dentro de los márgenes de separación que se hayan indicado. En este caso, estos márgenes tienen prioridad sobre el tamaño mínimo que se haya indicado para los Componentes, por lo que puede llegar a conseguirse que sean de un tamaño tan pequeño que sus etiquetas sean ilegibles.

Ejemplo de uso de GridLayout en un frame:



En este código debemos definir el gestor de diseño indicando fila y columnas.

El resultado de ejecutar el código anterior es



**GridBagLayout**

Esta clase pertenece al paquete java.awt y permite distribuir componentes en celdas que pueden o no ser de igual tamaño, formando filas y columnas.

Para poder utilizar este layout o gestor de diseño, se aconseja dibujar en papel primero para ver como deben quedar en el contenedor.

El correcto uso de este gestor de diseño va acompañado de los campos estáticos y públicos de la clase GridBagConstraint, estos campos permiten definir como y donde se deben añadir los diferentes componentes dentro del contenedor sobre un gestor GridBagLayout.

Algunos de los campos se detallan a continuación:

* **GridBagConstraints.gridx** nos dice la posición x del componente, es decir, el número de columna en la que está el componente, siendo la columna 0 la primera columna de la izquierda. Si el componente ocupa varias columnas (como nuestra área de texto), debemos indicar la columna en la que está la esquina superior izquierda del componente.
* **GridBagConstraints.gridy** nos dice la posición y del componente, es decir, el número de fila en la que está el componente, siendo la fila 0 la primera fila de la parte de arriba. Si el componente ocupa varias filas (como nuestra área de texto), debemos indicar la fila en la que está la esquina superior izquierda del componente.
* **GridBagConstraints.gridwidth** nos dice cuántas celdas en horizontal debe ocupar el componente. Representa el ancho del componente.
* **GridBagConstraints.gridheight** nos dice cuantas celdas en vertical debe ocupar el componente. Representa el alto del componente.

Ejemplo de uso de GridBagLayout:

 

El resultado de ejecutar este código es:



**NullLayout**

Cuando deseamos tener diseño propios en el contenedor y que no satisfacemos con algunos de los gestores de diseños vistos anteriormente, nos vemos obligados a definir el gestor como null, eliminándolo, y a hacernos cargo de la ubicación y tamaño de los componentes dentro del contenedor.

Para esto debemos utilizar cualquiera de los siguientes métodos con cada componente:

**setBounds (int x, int y, int width, int height):** este método permite definir la coordenada x, y donde se iniciará el componente, el ancho y el alto del componente, en pixels.

**setSize(int width, int height):** este método permite definir el ancho y el alto de un componente en pixels.

**setLocation(int x, int y):** este método permite definir la posición inicial del componentes en las coordinas x, y, esto en pixels.

Ejemplo de uso de null utilizando setBounds() para localizar los componentes:



Al ejecutar el código anterior se obtiene:



**BoxLayout**

Este gestor de diseño nace en la versión de swing.

Aquí los componentes son agrupados horizontal o verticalmente dentro del contenedor que los contiene. Los componentes no se sobreponen de ningún modo.

Este LayoutManager permite crear interfaces muy complejas así como nos permite GridBagLayout

El constructor de BoxLayout es muy simple:

public BoxLayout (Cont ainer o b j e t i v o , int e j e ) ;

Donde el parámetro entero eje puede tomar los valores:

• X\_AXIS, los componentes se organizan de izquierda a derecha y en horizontal.

• Y\_AXIS, los componentes se organizan de arriba a abajo y en vertical.

A continuación se presenta un ejemplo de uso de BoxLayout





Ejemplo de uso de BoxLayout.

La salida es:



Si se agranda la venta, esta se redimensiona:



**OverlayLayout**

Este gestor de diseño nace en la versión de swing.

OverlayLayout organiza los componentes en capas al estilo de CardLayout pero con varias diferencias:

• El tamaño de los componentes no es fijo sino que puede ser variable.

• El tamaño del contenedor será el tamaño del mayor de los componentes.

• Debido a que el tamaño de los componentes es diferente se pueden ver varios a la vez.

• A diferencia de CardLayout no existen métodos para navegar por los componentes.

Se dimensiona para contener el más grande de los Componentes y superpone cada Componente sobre los otros.

La clase **OverlayLayout** no tiene un constructor por defecto, así que hay que crearlo dinámicamente en tiempo de ejecución.

A continuación hay un ejemplo de uso de OverlayLayout centrando sus componentes.



El resultado de su ejecución es:



**GroupLayout**

En la version 6 del JDK aparece un nuevo LayoutManager,  se denomina GroupLayout y básicamente se centra en la creación anidada de grupos de componentes tanto horizontal (createSquentialGroup()) como verticalmente (createParallelGroup()), pudiendose añadir tanto grupos de componentes como componentes solamente.

Entre las facilidades que provee, permite fijar un espaciado automático entre componentes con setAutoCreateGaps(true) así­ como también establecer espaciado automático con respecto al contenedor (setAutoCreateContainerGaps()).

Esta clase se utiliza para crear complejas interfaces, preferentemente en Netbeans