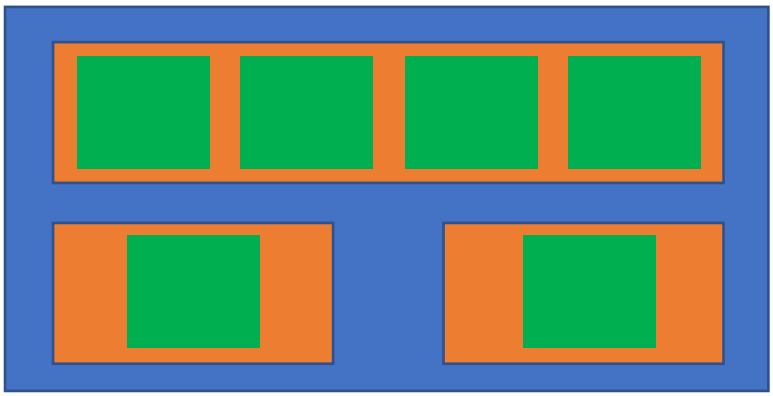
Disposiciones Avanzadas

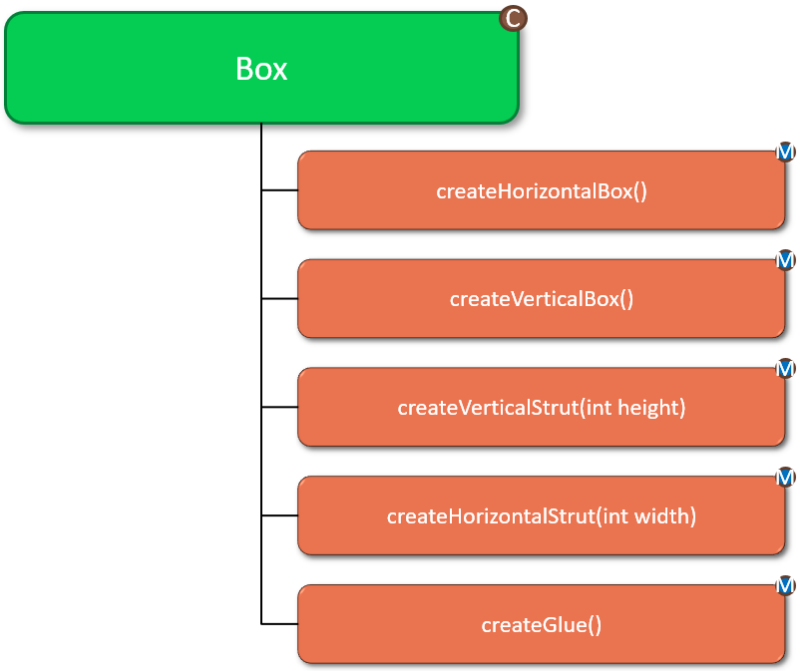
Disposición BOX (en caja):

Esta disposición basa su funcionamiento en la creación de contenedores con estructura de "cajas" (Box). Estos contendores-caja pueden estar anidados, es decir uno dentro de otro, en caso de que sea necesario para estructurar los elementos.



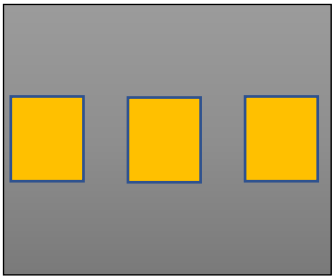
En esta imagen aparece reflejada la "filosofía" de este tipo de disposiciones Box. Cada rectángulo y cuadrado corresponde a un contenedor o "box" donde se pueden ubicar componentes swing.

Los contenedores box pueden estar en vertical u horizontal. Para construir este tipo de contenedores debemos utilizar la clase **Box** de la Api de Java. La clase Box cuenta con numerosos métodos en la API entre los que destacamos los siguientes:

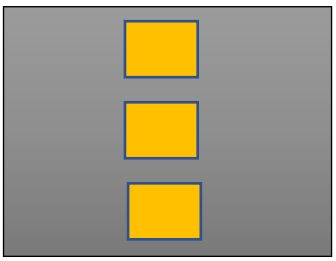


Clase Box y sus métodos más representativos

- **createHorizontalBox**: crea un contenedor "Box" que muestra los elementos de su interior en horizontal y de izquierda a derecha.



- **createVerticalBox:** crea un contenedor "Box" que muestra los elementos de su interior de arriba abajo con alineación horizontal centrada.



- **createVerticalStrut (int height):** crea un contenedor "Box" de altura fija con los elementos de su interior ajustados (a la altura disponible). El parámetro del método establece la separación vertical en píxeles entre cada uno de los elementos del interior.
- **createHorizontalStrunt (int width):** igual que el método anterior, pero con la disposición de los elementos del interior en horizontal.
- **createGlue():** crea una separación horizontal o vertical entre los elementos del interior de un Box capaz de separarse o juntarse dependiendo de las dimensiones del contenedor. Este método también es muy utilizado para separar contenedores entre si.

Disposición Spring (en muelle):

Cuando se emplea una disposición en muelle, se asocian muelles a cada componente. Estos muelles marcaran la separación que hay desde el componente hasta los elementos de su alrededor (otros componentes o bordes del contendor) y como se comporta dicha separación. La particularidad de este tipo de disposición, es que la separación entre los componentes se comporta como una especie de amortiguador o muelle capaz de comprimirse o separarse y donde podemos modificar diferentes parámetros.

Son dos las clases que necesitaremos para trabajar con la disposición Spring:

La clase **SpringLayout** es la encargada de determinar la disposición Spring del contenedor. El método **putConstraint()** es el encargado de especificar la separación entre componentes y su comportamiento:

putConstraint

```
public void putConstraint(String e1,
                          Component c1,
                          int pad,
                          String e2,
                          Component c2)
```

Links edge `e1` of component `c1` to edge `e2` of component `c2`, with a fixed distance between the edges. This constraint will cause the assignment

```
value(e1, c1) = value(e2, c2) + pad
```

to take place during all subsequent layout operations.

Parameters:

`e1` - the edge of the dependent

`c1` - the component of the dependent

`pad` - the fixed distance between dependent and anchor

`e2` - the edge of the anchor

`c2` - the component of the anchor

Método putConstraint() en la API

La clase **Spring** es la encargada de construir los "muelles" que separarán cada componente. El método **constant()** establece las características del muelle.

constant

```
public static Spring constant(int pref)
```

Returns a strut -- a spring whose *minimum*, *preferred*, and *maximum* values each have the value `pref`.

Parameters:

`pref` - the *minimum*, *preferred*, and *maximum* values of the new spring

Returns:

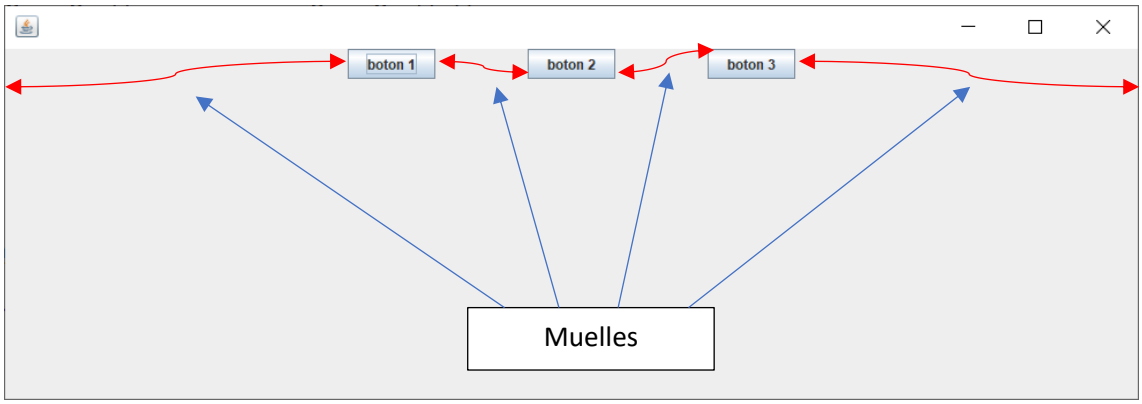
a spring whose *minimum*, *preferred*, and *maximum* values each have the value `pref`

Método constant() en la API

A continuación, se puede ver un ejemplo sencillo de disposición Spring:

```
package graficos;

import javax.swing.*;

public class DisposicionMuelle {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        MarcoMuelle mimarco=new MarcoMuelle();

        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    }

}

class MarcoMuelle extends JFrame{

    public MarcoMuelle(){

        setBounds(300,400,1000,350);

        LaminaMuelle milamina=new LaminaMuelle();

        add(milamina);

        setVisible(true);


    }

}
class LaminaMuelle extends JPanel{

    public LaminaMuelle(){


        JButton boton1=new JButton("boton 1");

        JButton boton2=new JButton("boton 2");

        JButton boton3=new JButton("boton 3");

        SpringLayout milayout=new SpringLayout();

        setLayout(milayout);

        add(boton1);

        add(boton2);

        add(boton3);

        Spring mimuelle=Spring.constant(0, 10, 100);

        Spring muelle_rigido=Spring.constant(80);

        milayout.putConstraint(SpringLayout.WEST, boton1, mimuelle, SpringLayout.WEST, this);

        milayout.putConstraint(SpringLayout.WEST, boton2, muelle_rigido, SpringLayout.EAST, boton1);

        milayout.putConstraint(SpringLayout.WEST, boton3, muelle_rigido, SpringLayout.EAST, boton2);

        milayout.putConstraint(SpringLayout.EAST, this, mimuelle, SpringLayout.EAST, boton3);



    }

}
```

Resultado de la ejecución del programa:



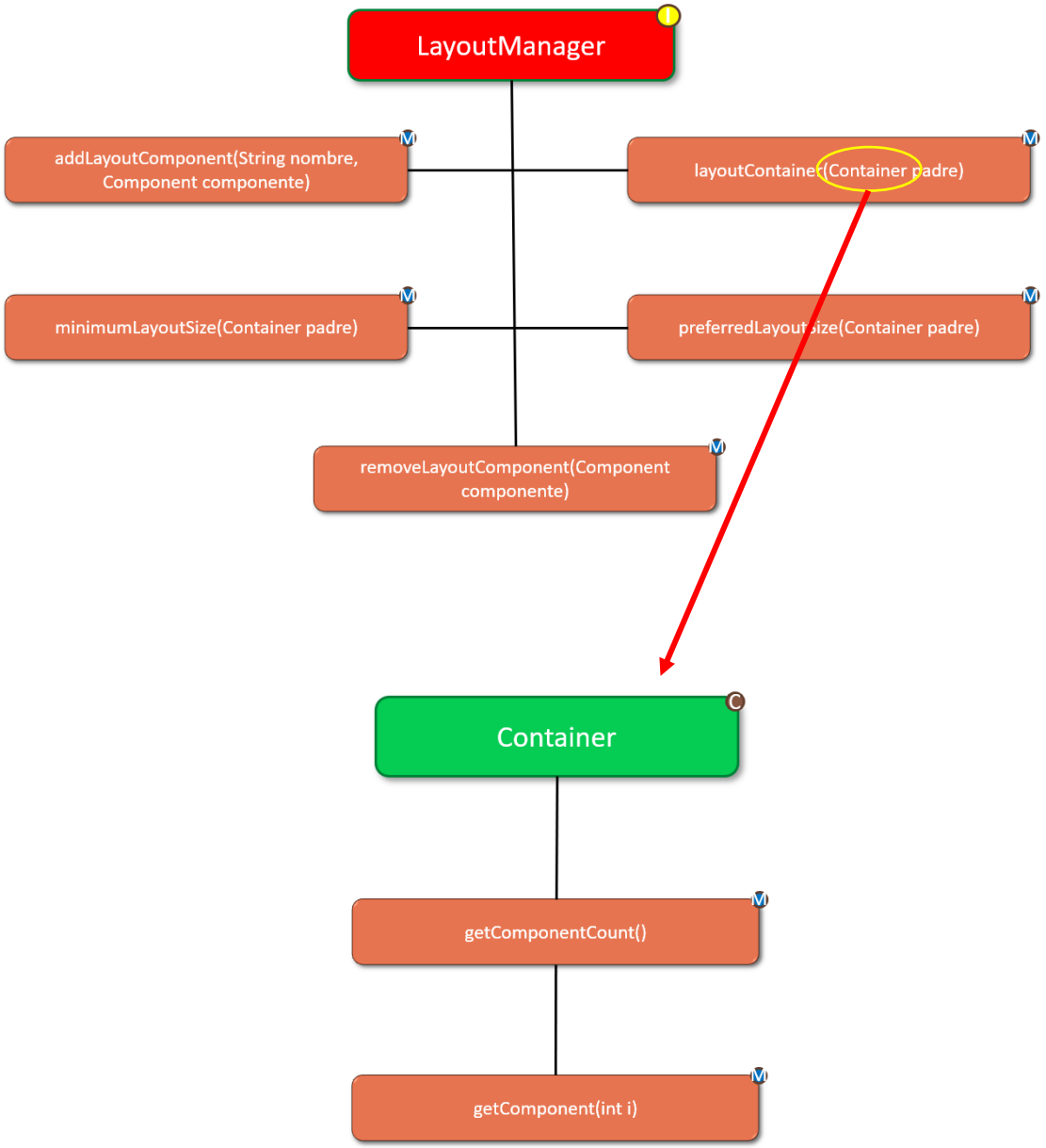
Disposición libre

Como su nombre indica, este tipo de disposición nos permite ubicar los componentes dentro del contendor en el lugar exacto donde queramos. Para ello se debe trabajar con el método **setBounds(x, y, width, height)** y las correspondientes coordenadas por ejemplo:

```
public LaminaDispLibre(){

    setLayout(null);

    JLabel nombre=new JLabel("Nombre: ");

    JLabel apellido=new JLabel("Apellido: ");

    JLabel tfno=new JLabel("Teléfono: ");

    JTextField c_nombre=new JTextField();

    JTextField c_apellido=new JTextField();

    JTextField c_tfno=new JTextField();

    nombre.setBounds(20, 20, 80, 10);

    c_nombre.setBounds(100, 15, 100, 20);

    apellido.setBounds(20, 60, 80, 10);

    c_apellido.setBounds(100, 55, 100, 20);

    add(nombre);

    add(c_nombre);
```

Como se observa en el código anterior, para trabajar con disposiciones libres lo primero que debemos hacer es especificar un layout nulo con la instrucción **setLayout(null)**. A continuación, se ubica cada elemento en las coordenadas indicadas en el método setBounds() con el tamaño indicado.

Sin embargo, esta forma de proceder no es la mejor a la hora de trabajar con disposiciones libres. Lo más útil es crear una clase propia encargada de construir la disposición libre para los componentes que queramos agregar al contenedor y son necesarias las siguientes interfaces, clases y métodos:



La interfaz **LayoutManager** tiene una lista de métodos en la que el más importante es el método **layoutContainer(Container parent).** Este método es el encargado de especificar cómo se va a construir la disposición libre recibiendo por parámetro un objeto de tipo **Container**. Este objeto

Container hace referencia al contenedor que albergará la disposición. Es importante por tanto conocer también la clase Container con sus métodos disponibles.

A continuación se muestra un ejemplo de creación de clase que implementa **LayoutManager** para la construcción de una disposición libre:

```
class EnColumnas implements LayoutManager{

    @Override
    public void addLayoutComponent(String arg0, Component arg1) {
        // TODO Auto-generated method stub

    }

    @Override
    public void layoutContainer(Container miContenedor) {
        // TODO Auto-generated method stub

        int d=miContenedor.getWidth();

        x=d/2;
        y=20;

        int n=miContenedor.getComponentCount();

        int contador=0;

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

            contador++;

            Component c=miContenedor.getComponent(i);

            c.setBounds(x-100, y, 100, 20);

            x+=100;


              if(contador%2==0) {

                  x=d/2;
                  y+=40;
              }

           }

    }

    @Override
    public Dimension minimumLayoutSize(Container arg0) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Dimension preferredLayoutSize(Container arg0) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public void removeLayoutComponent(Component arg0) {
        // TODO Auto-generated method stub

    }

    private int x=20;

    private int y=20;

}
```

El código del contenedor JPanel sería el siguiente:

```
    public LaminaDispLibre(){

        setLayout(new EnColumnas());

        JLabel nombre=new JLabel("Nombre: ");

        JLabel apellido=new JLabel("Apellido: ");

        JLabel tfno=new JLabel("Teléfono: ");

        JTextField c_nombre=new JTextField();

        JTextField c_apellido=new JTextField();

        JTextField c_tfno=new JTextField();

        add(nombre);

        add(c_nombre);

        add(apellido);

        add(c_apellido);

        add(tfno);

        add(c_tfno);


    }
}
```