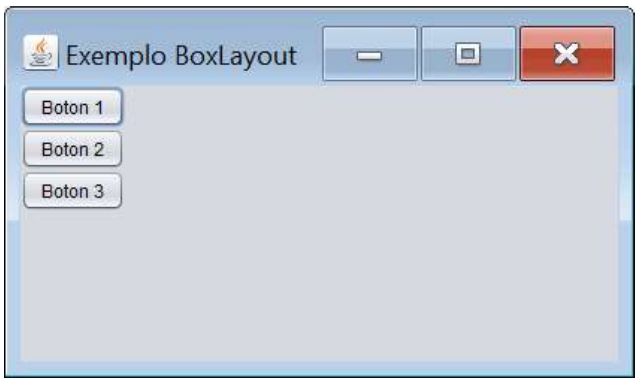
1.1.1.1 Layout manager BoxLayout

O layout manager BoxLayout defínese mediante a clase javax.swing.BoxLayout. O seu aspecto visual é o seguinte:

A imaxe anterior amosa un BoxLayout aplicado con disposición horizontal.



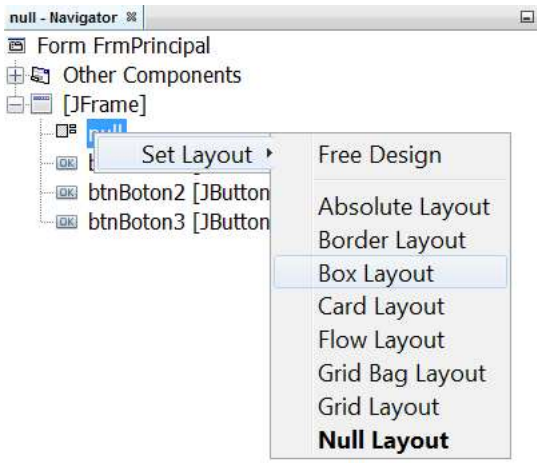
A imaxe anterior amosa un BoxLayout aplicado con disposición vertical. Nas imaxes anteriores amósase un contedor de tipo formulario sobre o que temos establecido un BoxLayout. Cando aplicamos un BoxLayout sobre un contedor os seus compoñentes sitúanse un tras outro (no caso de que definamos o BoxLayout como horizontal) ou apílanse un sobre outro (no caso de que definamos o BoxLayout como vertical). Para modificar o aliñamento e o tamaño de cada compoñente dentro do contedor sobre o que aplicamos o BoxLayout, debemos modificar certas propiedades dos compoñentes.

Configuración dun BoxLayout empregando NetBeans

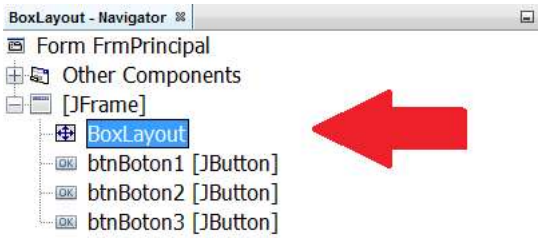
A continuación imos desenvolver o seguinte formulario:

Para empezar, creamos un proxecto e engadimos nel un formulario. Dentro do noso formulario engadimos tres botóns:

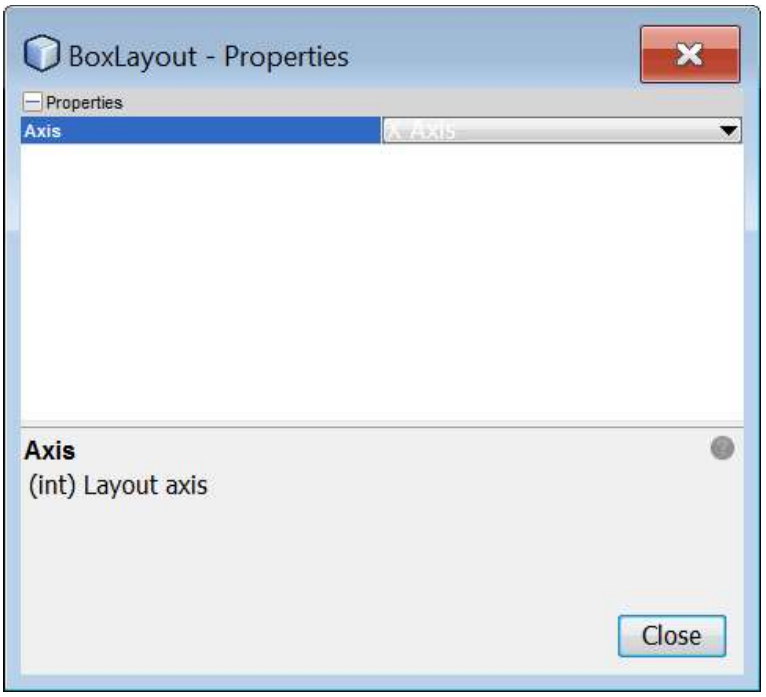
O seguinte que debemos facer é indicar que queremos establecer un layout manager de tipo BoxLayout sobre o noso formulario. Para isto prememos sobre o JFrame co botón dereito, eleximos a opción Set Layout e seleccionamos Box Layout:



A partir deste momento temos establecido para o noso formulario como layout manager un BoxLayout:

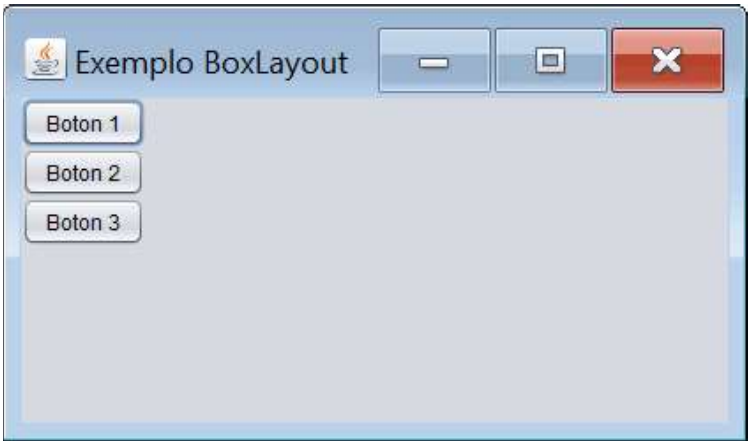


O seguinte paso é establecer se queremos amosar vertical ou horizontalmente os compoñentes dentro do noso xestor de distribución. Para facer isto accedemos ás propiedades do layout manager BoxLayout e modificamos a propiedade Axis dándolle o valor X Axis.



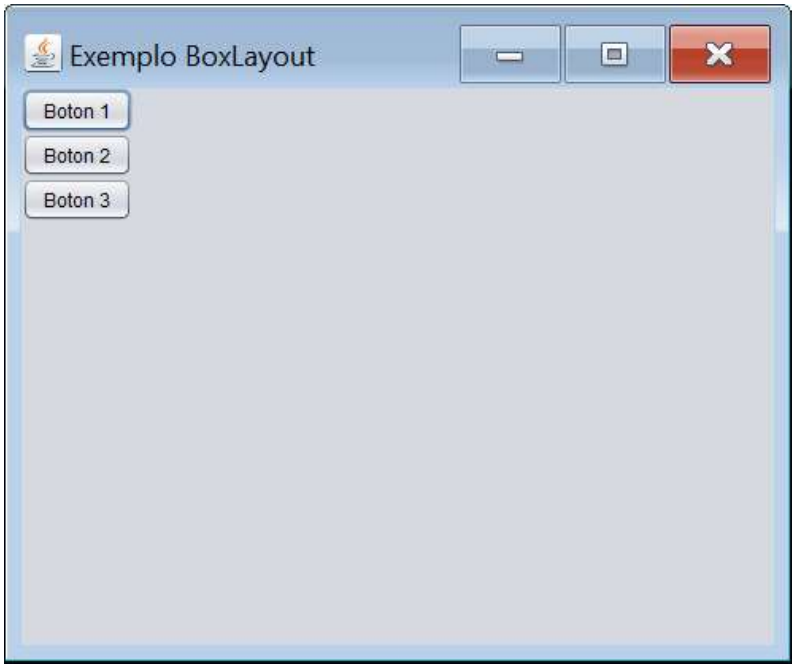
Como resultado obtemos o formulario coa distribución de compoñentes que perseguíamos:

Se queremos os compoñentes apilados verticalmente a única modificación que temos que facer é darlle á propiedade Axis do layout manager BoxLayout o valor Y Axis, co cal o resultado obtido será o seguinte:



Tamaño dos compoñentes

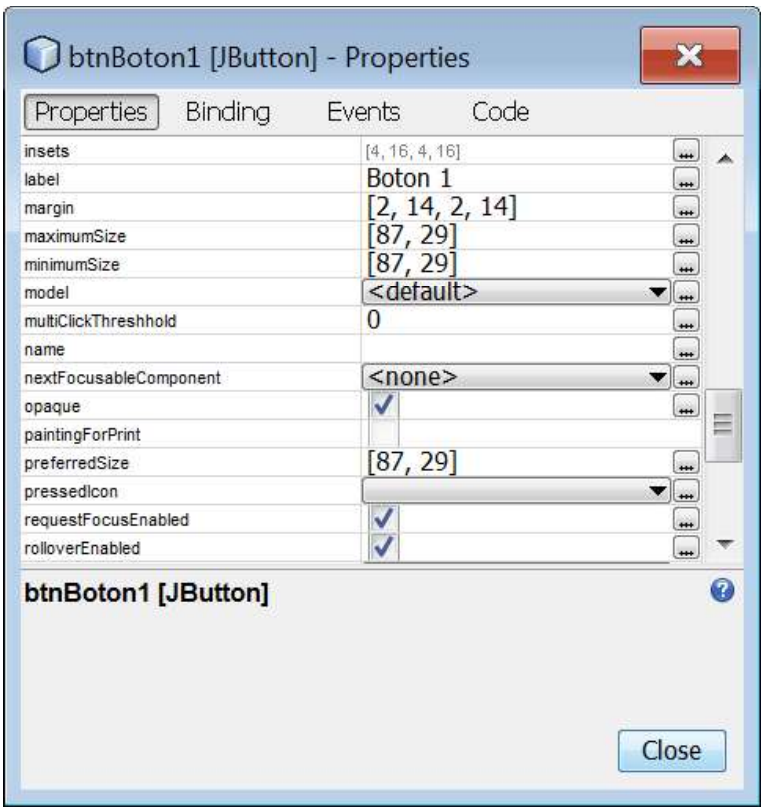
Ata agora o único que temos conseguido é aliñar os nosos compoñentes vertical e horizontalmente dentro do noso layout manager BoxLayout. Se atendemos aos exemplos anteriores pódese observar que aínda que modifiquemos o tamaño do contedor, os compoñentes continúan comportándose do mesmo xeito no tocante ao seu tamaño, e non se adecúan á nova configuración do contedor:



Este comportamento pode ser modificado. Para isto debemos xogar coas propiedades maximumSize, minimumSize e preferredSize dos compoñentes que son xestionados mediante un layout manager BoxLayout.

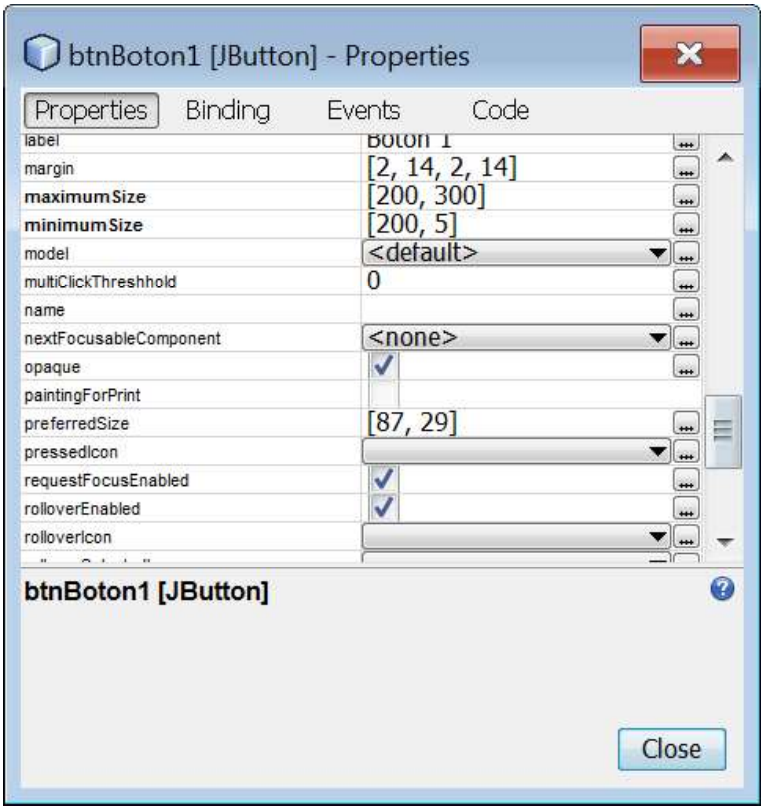
- A través da propiedade maximumSize establecemos o ancho e o alto (en píxels) máximo do noso compoñente.
- A través da propiedade minimumSize establecemos o ancho e o alto (en píxels) mínimo do noso compoñente.
- A través da propiedade preferredSize establecemos o ancho e o alto (en píxels) desexado para o noso compoñente.

Por defecto, para cada compoñente que engadamos no noso contedor (no caso de que este sexa xestionado por un layout manager de tipo BoxLayout) os valores destas propiedades son iguais:

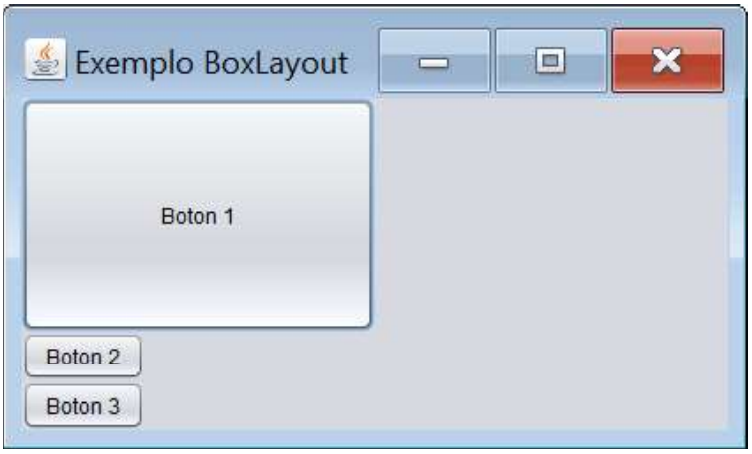


Empregando os valores por defecto estamos indicando que o compoñente sempre teña o mesmo tamaño. Aínda que redimensionemos o contedor, o compoñente sempre vai manter o seu tamaño fixo.

Supoñamos que queremos que o botón btnBoton1 poida ter un tamaño máximo de 200x300 píxels, un tamaño mínimo de 200x5 píxels, e un tamaño desexado de 87x29 píxels. Establecemos os valores indicados sobre as propiedades correspondentes:



O resultado é o seguinte:

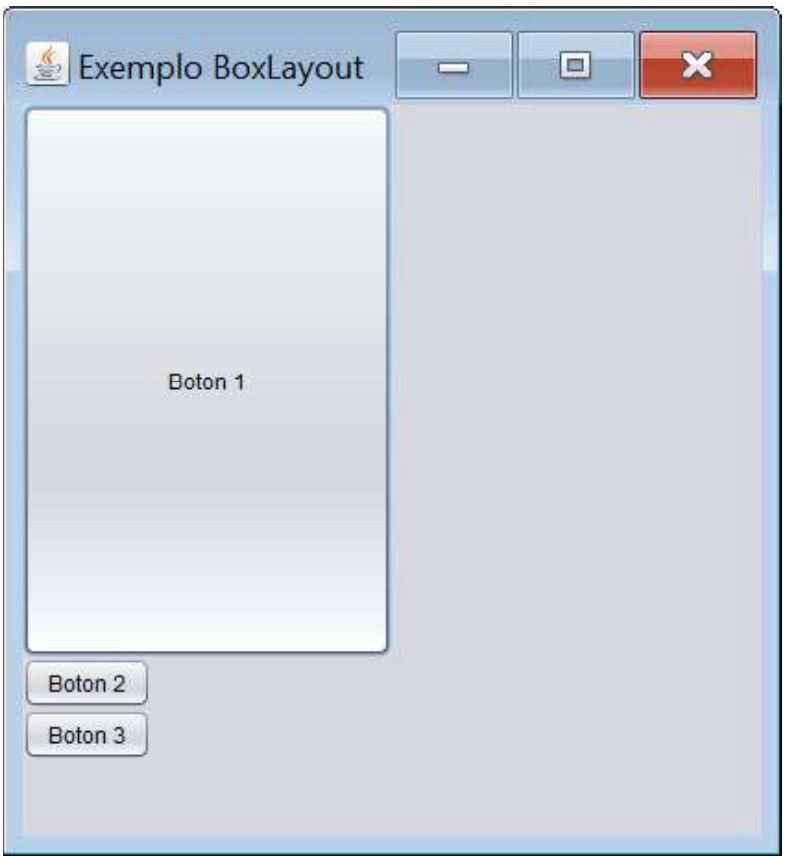


Para entender o acontecido temos que ter tamén en conta os valores das propiedades maximumSize, minimumSize e preferredSize dos outros compoñentes:

Compoñente	maximumSize	minimumSize	preferredSize
btnBoton1	200x300	200x5	87x29
btnBoton2	87x29	87x29	87x29
btnBoton3	87x29	87x29	87x29

Partindo desta información vemos que o btnBoton2 e o btnBoton3 miden o que teñen que medir (preferredSize), non obstante o btnBoton1 mide moito

máis do que debería. ¿Porque?. Ben, para o btnBoton1 indicamos que o seu preferredSize é de 87x29, pero atención, tamén indicamos que o seu maximumSize é de 200x300. Isto quere dicir o seguinte: o tamaño preferido do btnBoton1 é de 87x29, pero se hai espazo no contedor, o botón btnBoton1 pódese agrandar ata acadar un tamaño de 200x300, o indicado na súa propiedade maximumSize. Na imaxe anterior o contedor mide máis de 200 píxels de ancho, razón pola cal btnBoton1 estírase horizontalmente ata acadar os 200 píxels. Non obstante, btnBoton1 non se estira verticalmente ata acadar os 300 píxels senón que o layout manager BoxLayout só déixalle estirarse ata unha medida concreta. Esta medida está determinada polo tamaño vertical do contedor e polo preferredSize vertical dos outros compoñentes. Unha regra básica pola que se rexe o layout manager BoxLayout é a seguinte: hai que acomodar os compoñentes ao seu preferredSize, pero no caso de que exista máis espazo dispoñible no contedor, os compoñentes poderán aproveitar ese espazo extra agrandándose pero sen superar o seu maximumSize. Se agora agrandamos o tamaño do contedor este é o resultado:



Como pódese observar, o contedor ten espazo suficiente para acomodar os maximumSize de tódolos botóns e iso é o que fai. Na imaxe anterior a esta, como non había espazo suficiente primaban os preferredSize sobre os maximumSize.

Agora ben, ¿que ocorre se o que facemos é reducir o tamaño do formulario?. Pois ocorre algo bastante parecido, pero agora en lugar de fixarnos na propiedade maximumSize, debemos fixarnos na propiedade minimumSize. Supoñamos que reducimos o tamaño do formulario do seguinte

xeito:

Como pódese observar o contedor non ten espazo suficiente para acomodar os preferredSize dos botóns. Nesta situación o layout manager BoxLayout trata de acomodar os compoñentes empregando a propiedade minimumSize de cada un deles. Os botóns btnBoton2 e btnBoton3 amósanse co seu minimumSize (87x29) e xa non é posible que o seu tamaño sexa menor. Non obstante o minimumSize que fixamos para o botón btnBoton1 era menor (200x5) e por iso redúcese o seu tamaño ata acadar o seu minimumSize. Na seguinte imaxe pódese observar que unha vez acadados os minimumSize de cada compoñente, se seguimos reducindo o tamaño do contedor perderemos visibilidade sobre os compoñentes:

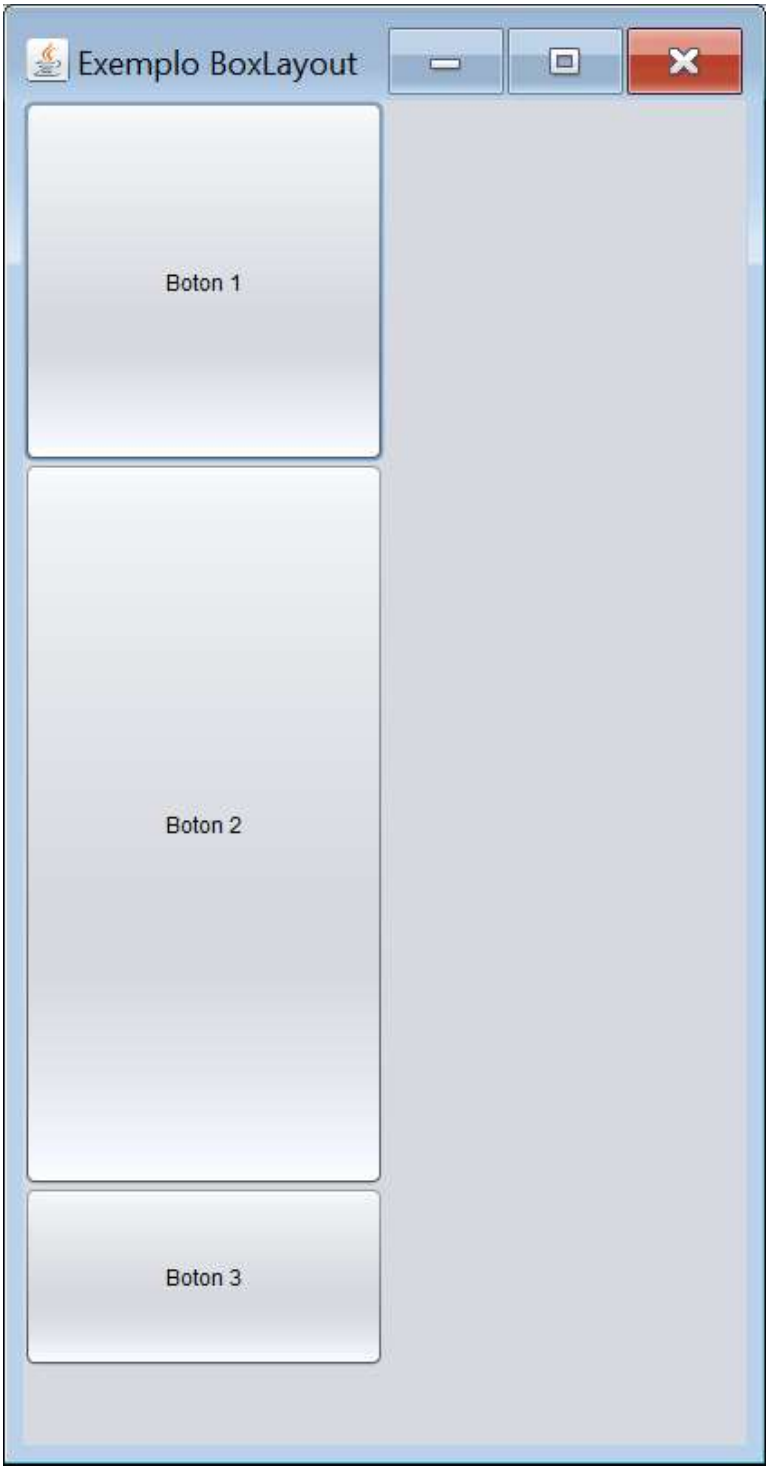
Outra regra básica que rexe o comportamento do layout manager BoxLayout é a seguinte: hai que acomodar os compoñentes ao seu preferredSize, pero no caso de que non exista suficiente espazo os compoñentes poderán reducir o seu tamaño sempre que este non sexa menor que o seu minimumSize.

Proporcionalidade

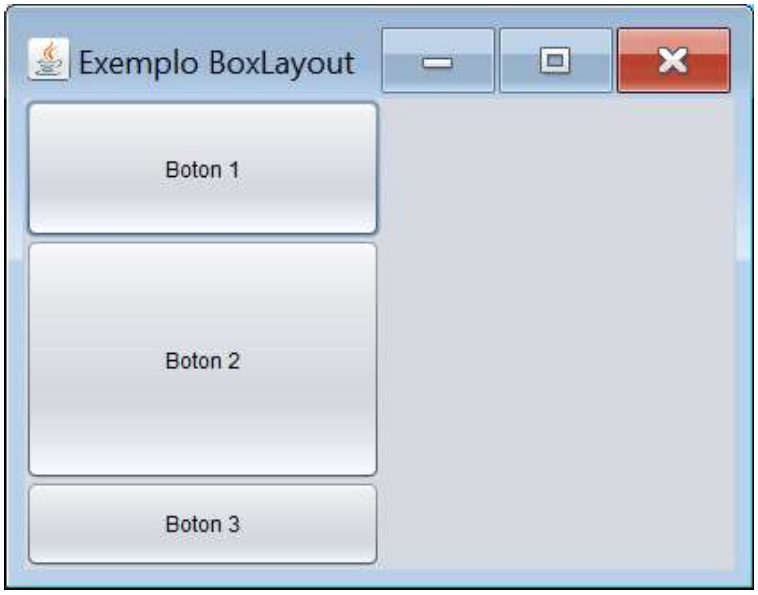
No exemplo anterior unicamente varía o tamaño dun dos compoñentes (btnBoton1) ao redimensionar o formulario. ¿Que ocorre cando máis dun compoñente pode modificar o seu tamaño? ¿Como é repartido o espazo entre eles?. Supoñamos que establecemos as seguintes medidas sobre os compoñentes do exemplo anterior:

Compoñente	maximumSize	minimumSize	preferredSize
btnBoton1	200x200	200x5	87x29
btnBoton2	200x400	200x5	87x29
btnBoton3	200x100	200x5	87x29

No caso de que exista suficiente espazo non hai problema:



Aplícase o maximumSize de cada compoñente. Pero, ¿que ocorre se non hai espazo suficiente para aplicar o maximumSize? Imaxinemos que redimensionamos o formulario do seguinte xeito:



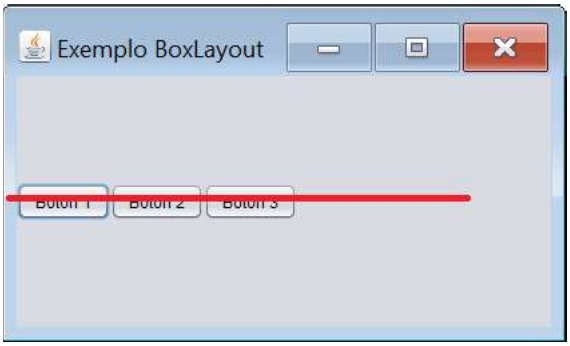
Non hai altura suficiente para que os botóns empreguen os seus maximumSize (verticais) correspondentes. Polo tanto, os botóns teñen que repartirse o espazo, e para iso establecen entre eles unha relación de proporcionalidade baseada no seu maximumSize. Aproximadamente, o calculo que realiza o layout manager BoxLayout é o seguinte:

Compoñente	maximumSize	% de altura a tomar no caso de que non exista suficente espazo
btnBoton1	200x200	±28
btnBoton2	200x400	±57
btnBoton3	200x100	±14

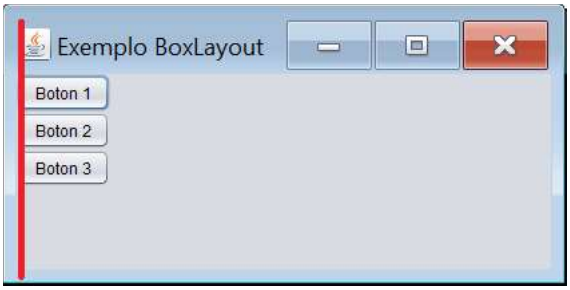
Entre os 3 botóns, requiren de 700 píxels (verticais) para poder aplicar os seus maximumSizes (200+400+100). O que fai o layout manager BoxLayout é distribuír o espazo existente entre os compoñentes proporcionalmente aos seus requirimentos de maximumSize (p.e., para o botón btnBoton2 precisa de 400 píxels. Isto é aproximadamente o 57% de 700 píxels. Polo tanto, no caso de que non exista espazo suficiente, correspóndelle aproximadamente o 57% do espazo existente).

Aliñamento dos compoñentes

Independentemente do tamaño dos compoñentes do contedor sobre o que aplicamos o layout manager BoxLayout, podemos observar que ata o momento sempre ocorre que cando empregamos o layout manager BoxLayout horizontal os compoñentes están centrados verticalmente respecto ao contedor. Na seguinte imaxe a liña vermella indica o aliñamento:



Cando empregamos o layout manager BoxLayout vertical os compoñentes están aliñados á esquerda respecto ao contedor. Na seguinte imaxe a liña vermella indica o aliñamento:



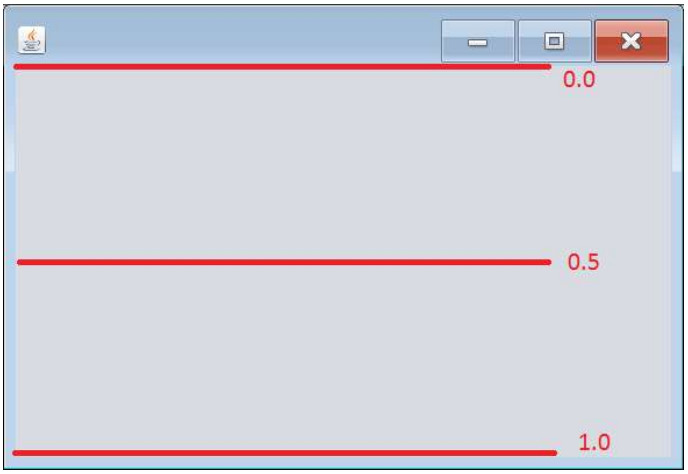
Para cambiar o aliñamento dos compoñentes respecto ao contedor, debemos modificar a seguinte propiedade do compoñente:

- Propiedade alignmentY no caso de que a distribución dentro do BoxLayout sexa horizontal.
- Propiedade alignmentX no caso de que a distribución dentro do BoxLayout sexa vertical.

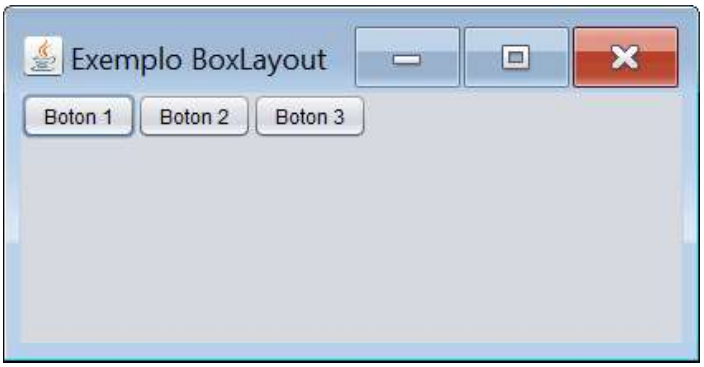
Propiedade alignmentY. Mediante esta propiedade modificamos o aliñamento vertical do compoñente respecto ao contedor. O valor que pode tomar esta propiedade é un número decimal entre 0.0 e 1.0. Os valores empregados habitualmente son os seguintes:

- 0.0: aliñamento superior do compoñente respecto ao contedor.
- 0.5: aliñamento centrado do compoñente respecto ao contedor (comportamento por defecto).
- 1.0: aliñamento inferior do compoñente respecto ao contedor.

Na seguinte imaxe pódese observar como aliñéanse os compoñentes en función do valor da propiedade alignmentY:



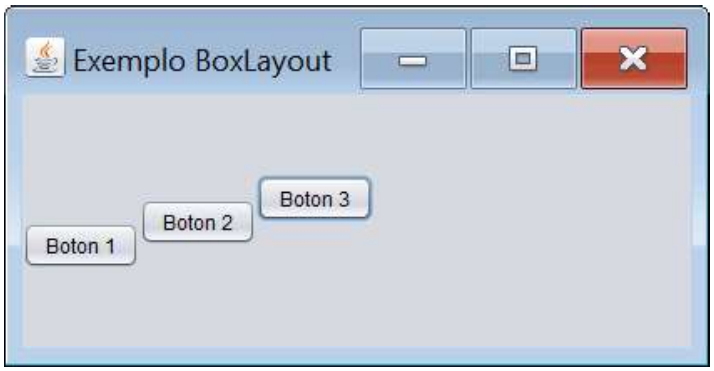
No seguinte exemplo establécese o valor de alignmentY a 0.0 para tódolos compoñentes:



Neste outro exemplo establécese o valor de alignmentY a 0.5 para tódolos compoñentes (comportamento por defecto):

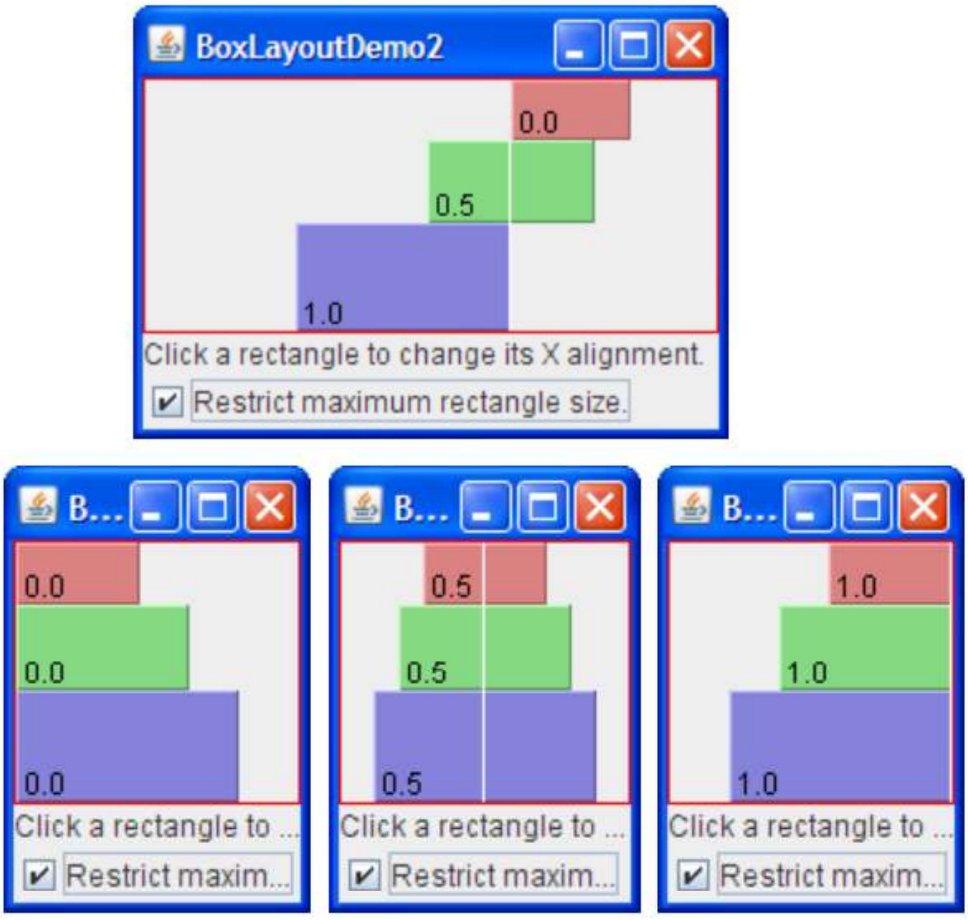
Por último, neste exemplo establécese o valor de alignmentY a 1.0 para tódolos compoñentes:

O máis habitual é que tódolos compoñentes do contedor teñan o mesmo valor de alignment Y, aínda que isto non é obrigatorio. No caso de que empreguemos diferentes valores pódense conseguir efectos do máis curiosos, p.e.: se modificamos o exemplo sobre o que estamos traballando de xeito que a distribución do layout manager BoxLayout sexa horizontal e a propiedade alignmentY dos botóns btnBoton1, btnBoton2 e btnBoton3 valga 0.0, 0.5 e 1.0 respectivamente, o resultado sería o seguinte:



Propiedade alignmentX. Mediante esta propiedade modificamos o aliñamento horizontal dos compoñentes respecto ao contedor. O valor que pode tomar esta propiedade é un número decimal entre 0.0 e 1.0. Os valores empregados habitualmente son os seguintes:

- 0.0: aliñamento á esquerda do compoñente respecto ao contedor (comportamento por defecto).
- 0.5: aliñamento centrado do compoñente respecto ao contedor.
- 1.0: aliñamento á dereita do compoñente respecto ao contedor.

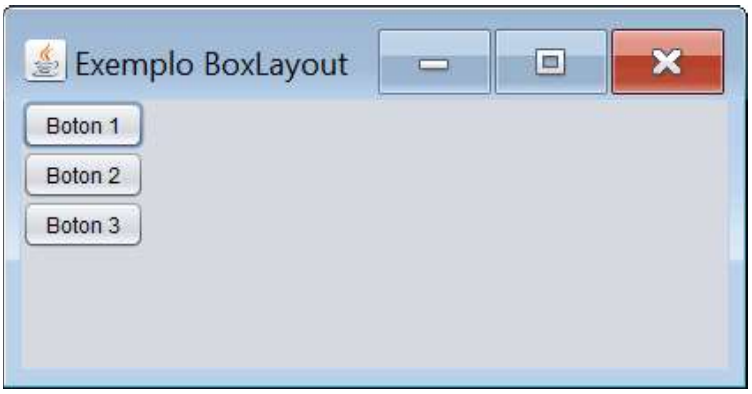


Aclaración ao comportamento do Box Layout que pode verse claro nas dúas imaxes anteriores. Toda a documentación na seguinte ligazón:

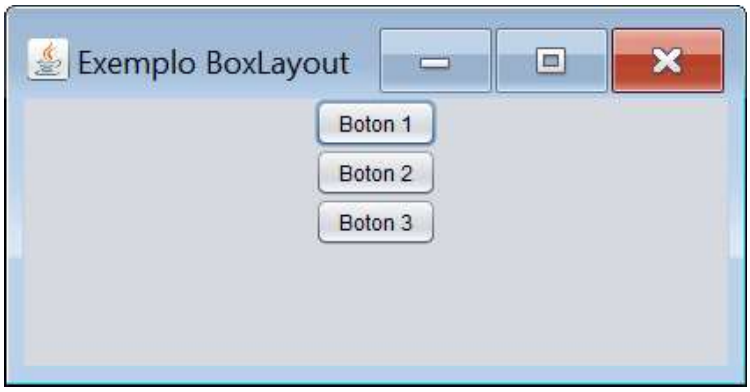
https://docs.oracle.com/javase/tutorial/uiswing/layout/box.html

Na seguinte imaxe pódese observar como se aliñean os compoñentes en función do valor da propiedade alignmentX:

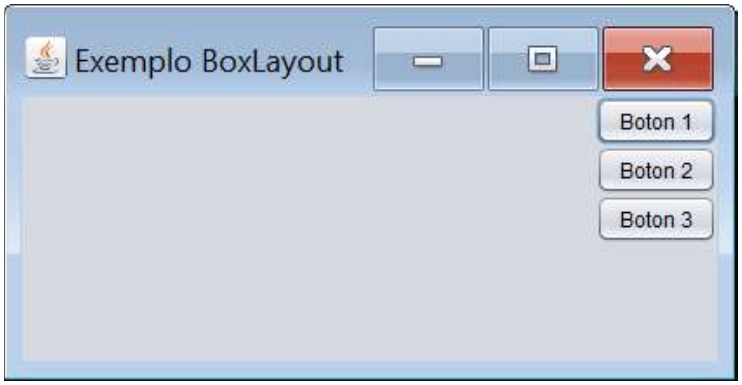
No seguinte exemplo establécese o valor de alignmentX a 0.0 para tódolos compoñentes (comportamento por defecto):



Neste outro exemplo establécese o valor de alignmentX a 0.5 para tódolos compoñentes:

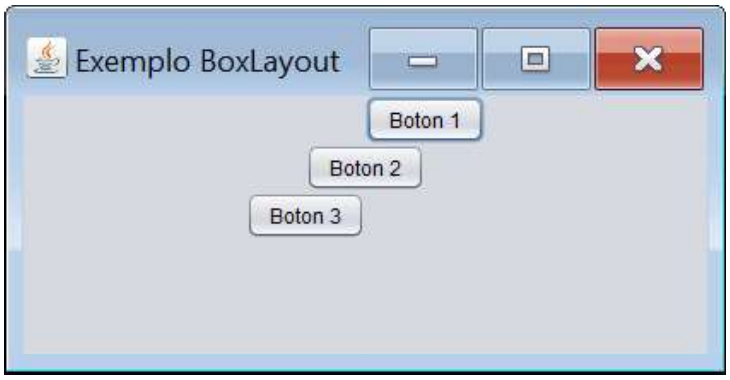


Por último, neste exemplo establécese o valor de alignmentX a 1.0 para tódolos compoñentes:



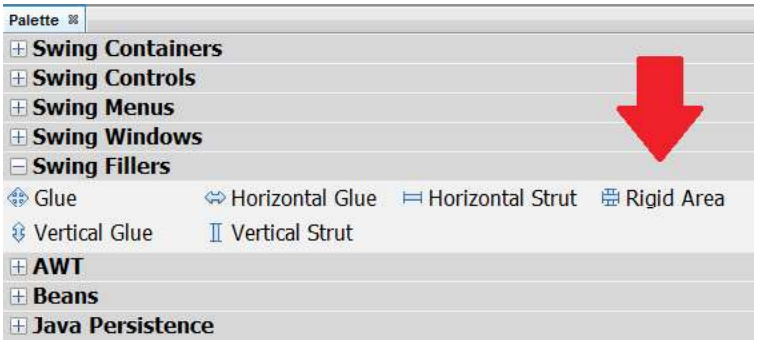
O máis habitual é que tódolos compoñentes do contedor teñan o mesmo valor de alignment X, aínda que isto non é obrigatorio. No caso de que empreguemos diferentes valores pódense conseguir efectos do máis curiosos, p.e.: se modificamos o exemplo sobre o que estamos traballando de xeito que a distribución do layout manager BoxLayout sexa vertical e a propiedade

alignmentX dos botóns btnBoton1, btnBoton2 e btnBoton3 valga 0.0, 0.5 e 1.0 respectivamente, o resultado sería o seguinte:



Recheadores

Quédanos por resolver un problema á hora de traballar co layout manager BoxLayout. Este problema consiste en como separar uns compoñentes doutros. Ata agora os nosos compoñentes encontráronse sempre practicamente pegados uns aos outros. Apenas uns poucos píxels separaban a uns dos outros. Para solucionar este problema empregamos compoñentes da clase Filler (javax.swing.Box.Filler). Os obxectos que pertencen a esta clase teñen a característica de que visualmente non debuxan nada no contedor. Son caixas transparentes. A pesar diso, ocupan un alto e un ancho e polo tanto a súa funcionalidade é a de crear un oco entre os compoñentes que separan. As normas referentes a maximumSize, minimumSize, preferredSize, alignmentX e alignmentY aféctalles exactamente igual que ao resto de compoñentes, coa peculiaridade de que estes compoñentes o que debuxan no contedor é un oco. Para empregar este compoñente seleccionamos na paleta de compoñentes o compoñente Rigid Area:

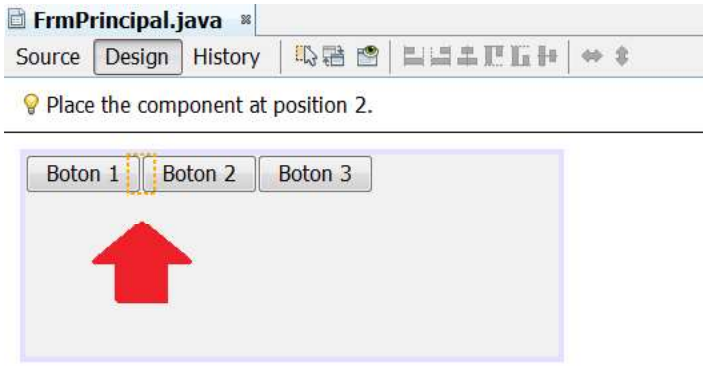


Ao seleccionalo, un pequeno asistente pregúntanos acerca do ancho e do alto do compoñente:

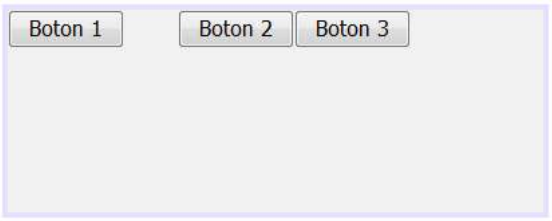
Unha vez establecido o tamaño do Filler, unicamente queda arrastralo a o seu lugar, que será entre os dous compoñentes que desexamos separar. Por último e xa opcionalmente, poderemos acceder ás propiedades do Filler e modificar o seu maximumSize, minimumSize, preferredSize, alignmentX e alignmentY de xeito que adecúese aos nosos requirimentos visuais. Por exemplo, dado o seguinte formulario:

se queremos establecer unha separación de 40 píxels entre o Boton 1 e o Boton 2 faremos o seguinte: na paleta de compoñentes prememos sobre o compoñente Rigid Area. Abrirásenos o asistente deste compoñente e nel indicamos que queremos que o ancho do compoñente sexa de 40 píxels:

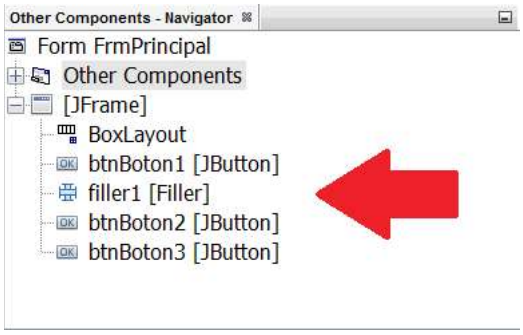
De seguido, prememos en OK e arrastramos o compoñente ao seu lugar de destino, neste caso, entre o Boton 1 e o Boton 2:



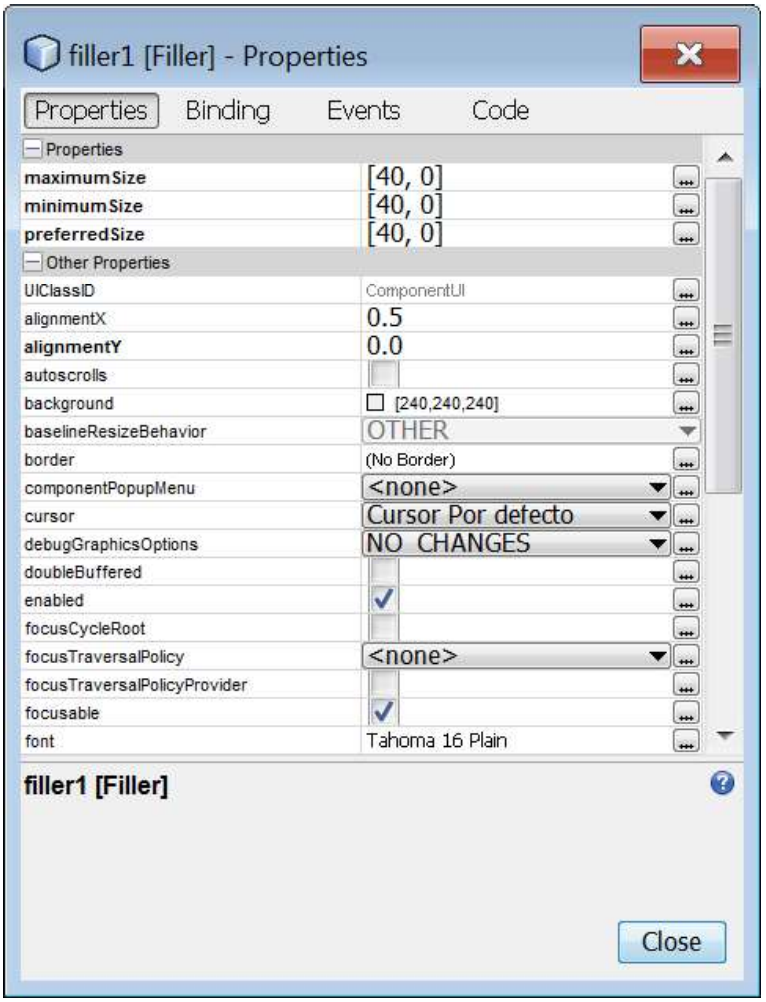
Unha vez situado o Filler no seu lugar, este é o resultado na xanela de deseño:



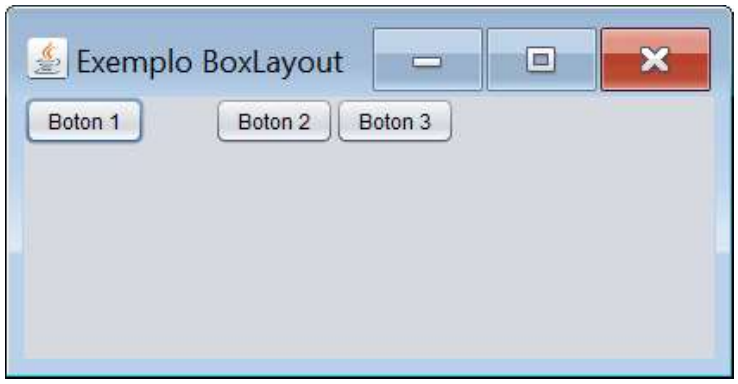
Como pódese observar os botóns 1 e 2 están separados. O compoñente que os separa é o Filler que acabamos de colocar. Fixándonos na árbore de compoñentes do noso proxecto veremos que hai un novo compoñente de tipo Filler. Este compoñente é o Filler que acabamos de engadir.



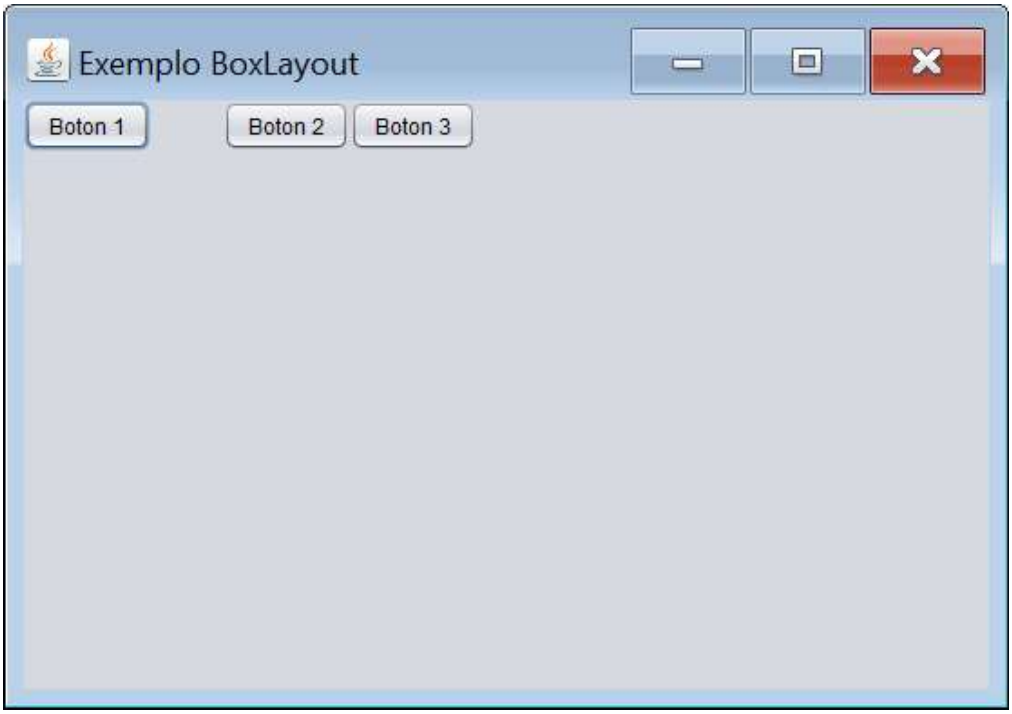
Por ultimo, se en calquera momento quixeramos modificar as características de tamaño ou de aliñación deste Filler, podemos acceder ás súas propiedades e modificalas:



O resultado da execución do noso formulario empregando un Filler é o seguinte:



Como pódese apreciar, os compoñentes Boton 1 e Boton 2 están separados por unha distancia de 40 píxels xerada mediante o emprego dun Filler. No caso de que redimensionemos a xanela, esta separación entre os botóns mantense:



Os Fillers non se empregan unicamente dentro dun xestor de distribución de tipo BoxLayout. Sempre que queiramos separar dous compoñentes entre si, podemos empregar un Filler para crear un oco entre eles.

A continuación amósase o listado do código fonte desenvolvido para a realización da aplicación (dentro da carpeta dos anexos referente a esta actividade pódese atopar a carpeta Actividades\BOXLAYOUT, a cal contén un proxecto NetBeans referente á explicación que acabamos de rematar). O código amosado corresponde ao último exemplo desenvolvido:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package exemploBoxLayout;

/**
 *
 * @author DAM2
 */
public class FrmPrincipal extends javax.swing.JFrame {

    /**
     * Creates new form FrmPrincipal
     */
    public FrmPrincipal() {
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        btnBoton1 = new javax.swing.JButton();
        filler1 = new javax.swing.Box.Filler(new java.awt.Dimension(40, 0), new java.awt.Dimension(40,
0), new java.awt.Dimension(40, 0));
```

```
        btnBoton2 = new javax.swing.JButton();
        btnBoton3 = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Exemplo BoxLayout");
        getContentPane().setLayout(new                          javax.swing.BoxLayout(getContentPane(),
javax.swing.BoxLayout.X_AXIS));

        btnBoton1.setText("Boton 1");
        btnBoton1.setActionCommand("");
        btnBoton1.setAlignmentY(0.0F);
        getContentPane().add(btnBoton1);

        filler1.setAlignmentY(0.0F);
        getContentPane().add(filler1);

        btnBoton2.setText("Boton 2");
        btnBoton2.setActionCommand("");
        btnBoton2.setAlignmentX(0.5F);
        btnBoton2.setAlignmentY(0.0F);
        getContentPane().add(btnBoton2);

        btnBoton3.setText("Boton 3");
        btnBoton3.setActionCommand("");
        btnBoton3.setAlignmentX(1.0F);
        btnBoton3.setAlignmentY(0.0F);
        getContentPane().add(btnBoton3);

        java.awt.Dimension screenSize = java.awt.Toolkit.getDefaultToolkit().getScreenSize();
        setBounds((screenSize.width-422)/2, (screenSize.height-216)/2, 422, 216);
    }// </editor-fold>

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for               (javax.swing.UIManager.LookAndFeelInfo               info               :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(FrmPrincipal.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(FrmPrincipal.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(FrmPrincipal.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(FrmPrincipal.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        }
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new FrmPrincipal().setVisible(true);
            }
        });
    }
    // Variables declaration - do not modify
    private javax.swing.JButton btnBoton1;
    private javax.swing.JButton btnBoton2;
    private javax.swing.JButton btnBoton3;
    private javax.swing.Box.Filler filler1;
    // End of variables declaration
}
```