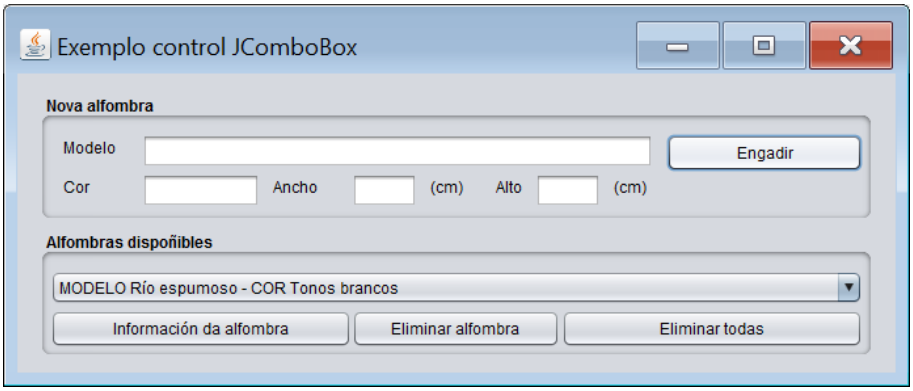
1. Compoñente Combo (JComboBox)

O compoñente combo conceptualmente é moi parecido ao compoñente lista. O combo é un compoñente que conten unha serie de posibles valores dos cales o usuario pode elixir un. Visualmente pode presentar dous estados: un primeiro estado reducido no cal o seu aspecto é parecido a unha caixa de texto e amosa o elemento seleccionado e un segundo estado no cal respondendo a unha pulsación do usuario sobre unha zona concreta do combo, desprégase unha lista na cal amósanse os valores posibles que poden seleccionarse para o compoñente.

O tamaño ocupado por un combo nun contedor é moito máis reducido que o ocupado por unha lista, xa que ocupa unicamente o espazo que ocuparía unha caixa de texto sendo esta unha das razóns para empregar combos en lugar de listas. A información que conten un combo pode ser modificada dinamicamente en función de distintos eventos. A xestión dos combos en java fai uso do paradigma vista controlador visto con anterioridade. O compoñente combo defínese a través da clase javax.swing.JComboBox. Graficamente o seu aspecto é o seguinte:



Na imaxe anterior podemos ver un combo que conten unha serie de valores. Visualmente unicamente pódese visualizar o valor seleccionado, non obstante o compoñente permite ao usuario despregar unha lista que conten outros valores posibles para o combo e realizar unha nova selección.

Propiedades do control JComboBox empregadas máis habitualmente e que pódense establecer a través do entorno de programación:

Propiedade	Función
Background	Cor de fondo do compoñente
Border	Borde do compoñente
Cursor	Mediante esta propiedade indícase a imaxe que amosará o punteiro do rato ao navegar sobre o compoñente
Enabled	Se esta propiedade está activada o compoñente será funcional. No caso de que esta propiedade estea desactivada o compoñente será visualizable, pero o usuario non poderá interaccionar con el.

Focusable	Se esta propiedade está activada o compoñente entrará na roda de reparto do foco de xeito que ao premer a tecla de cambio de foco (habitualmente o tabulador) nalgún momento tomará o foco da aplicación (cando sexa a súa quenda na roda de reparto do foco). Pola contra, se esta propiedade está desactivada o compoñente non entrará na roda de reparto do foco e soamente será posible acceder a el mediante o rato.
Font	Características da fonte do compoñente (tipo de fonte, tamaño, ...)
Foreground	Cor do texto do compoñente
Model	Modelo de datos. Estrutura de datos que conten a información que é gardada no combo
SelectedIndex	Índice do elemento seleccionado no combo (o primeiro é o 0). Con -1 indicamos que non hai ningún elemento seleccionado.
ToolTipText	Mediante esta propiedade establécese unha mensaxe (tooltip) que será amosado ao deixar o punteiro do rato sobre o compoñente. É habitual empregar esta mensaxe para indicar cal é a utilidade do compoñente

Eventos do control JComboBox empregados máis habitualmente e que pódense establecer a través do entorno de programación:

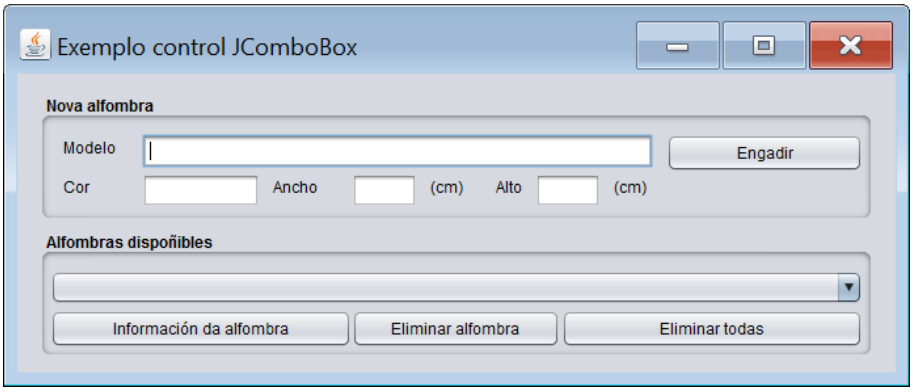
Evento	Lanzamento
ItemStateChanged	Este evento é disparado cando cambiamos o estado dun elemento do combo. Este evento pode dispararse en dúas situacións: a primeira situación dáse cando deseleccionamos un elemento e a segunda situación dase cando seleccionamos un elemento. O evento prové información acerca de se foi disparado por unha selección ou por unha deselección, sendo tarefa do programador adecuar o código á situación acontecida.

Métodos do control JComboBox empregados máis habitualmente:

Método	Función
public int getSelectedIndex()	Devolve o índice do elemento seleccionado no combo (o primeiro elemento é o 0). Devolve -1 se non hai ningún elemento seleccionado.
public void setSelectedIndex(int anIndex)	Establece o elemento seleccionado no combo a través do seu índice (o primeiro elemento é o 0). Se pasamos un -1 indicamos que non hai ningún elemento seleccionado.

Xestión de combos empregando NetBeans (primeiro exemplo)

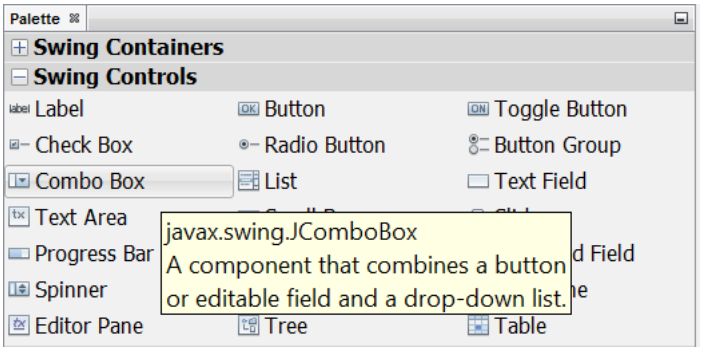
A continuación imos desenvolver un formulario bastante parecido a un dos desenvolvidos para a explicación do compoñente lista, pero substituíndo a lista por un combo. Este será o aspecto do formulario:



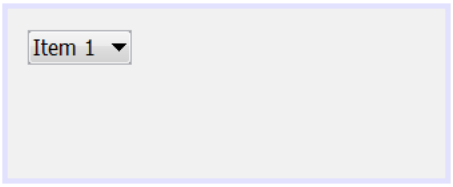
Como pódese observar cando substituímos a lista por un combo o tamaño do formulario redúcese de forma significativa. Neste caso, o formulario consta dun combo, o cal atópase no panel de alfombras dispoñibles. Emprégase para engadir nel as alfombras que definamos desde o panel nova alfombra. A través

dos tres botóns adxuntos ao combo podemos realizar diferentes accións sobre os elementos do combo. Neste caso non detectaremos o dobre clic sobre o combo xa que é unha acción de usuario que non ten sentido (aínda pode ser detectado). As caixas de texto Ancho e Alto só admitirán números e a súa lonxitude estará limitada a cinco caracteres. Calquera erro que poida acontecer ao longo do emprego do programa debe ser reportado ao usuario.

Para engadir un compoñente de tipo JComboBox no noso formulario primeiramente seleccionarémolo da paleta de compoñentes do entorno de programación:



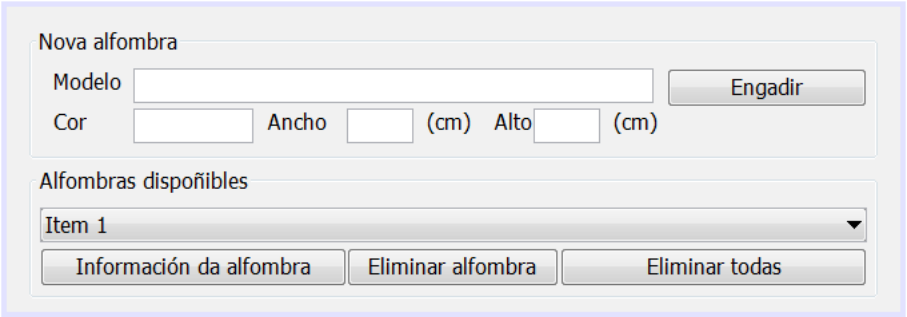
e arrastrámolo ata o formulario:



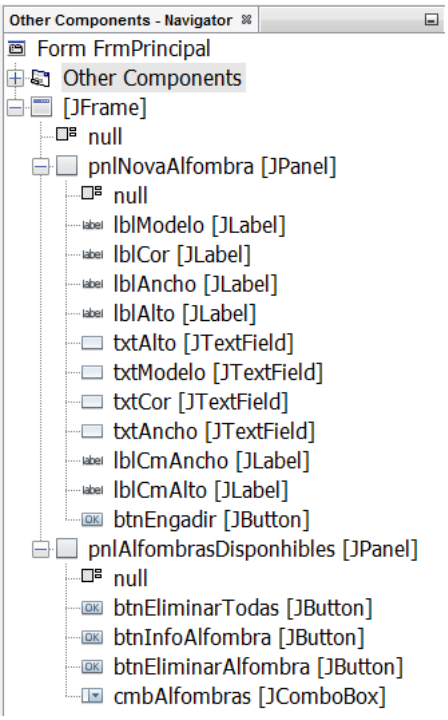
Ao contrario que ocorría coa lista, o combo ten a capacidade de proporcionar ao usuario unha barra de desprazamento para acceder a aqueles elementos que conten e que non son visualizables.

Unha vez situado o compoñente dentro do formulario, adaptámolo para que teña o aspecto visual que desexamos que teña.

No caso que estamos desenvolvendo necesitamos engadir seis etiquetas, catro caixas de texto, catro botóns, dous paneis e un combo. Unha vez engadidos e colocados no seu lugar correspondente, este será o resultado do deseño:

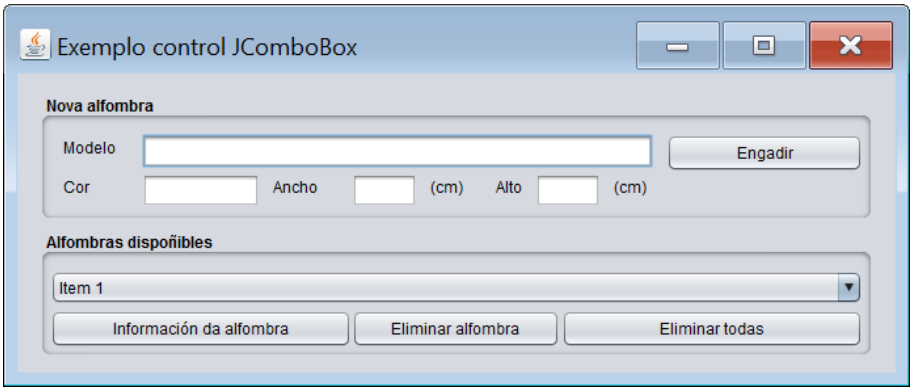


O noso formulario quedará configurado do seguinte xeito:



Unha vez que temos colocados os compoñentes que necesitamos hai que configurar o seu aspecto, pero neste caso os valores que teñen por defecto os compoñentes son os axeitados para os nosos requirimentos

Se executamos a aplicación este será o resultado:



Como pódese observar o comportamento visual da aplicación xa está resolto. Agora imos desenvolver a súa funcionalidade.

O funcionamento do combo baséase no paradigma vista controlador (ver compoñente lista). O obxecto que empregamos como almacén de datos para almacenar os obxectos que se amosan no combo asociado a ese almacén de datos, será un obxecto da clase javax.swing.DefaultComboBoxModel. O seu funcionamento é idéntico ao visto para la clase javax.swing.DefaultDefaultListModel. Permite xestionar dinamicamente o seu contido (engadir obxectos, eliminalos, modificalos, ...). Neste caso imos declarar o almacén de datos como un atributo da clase, de xeito que poidamos

acceder a el desde calquera método do formulario:

```
private javax.swing.DefaultComboBoxModel modeloAlfombras;
```

Como pódese observar, neste caso non tipamos o obxecto tal e como fixemos co almacén de datos da lista. Neste caso, ao non tipar o almacén de datos, estamos indicando que admite obxectos de calquera clase. Elo implica que á hora de recuperar os obxectos gardados no almacén de datos deberemos facerlles un cast para convertelos ao que realmente son. Sería posible tipar o almacén de datos, pero non o faremos para ter as dúas perspectivas da resolución do problema (tipado na lista e sin tipar no combo).

Unha vez definida a estrutura de datos que vai funcionar como almacén dos datos do combo, temos que ligala ao compoñente gráfico de tipo combo que vai amosar os seus datos. Isto o facemos nalgún punto do código que se execute antes de que o usuario poida empezar a interaccionar co formulario. Quizás, o punto ideal é no construtor do formulario e despois da chamada que realiza para pintar os compoñentes gráficos do formulario:

```
    public FrmPrincipal() {
        initComponents();
        //crear o modelo para almacenar as alfombras
        modeloAlfombras=new DefaultComboBoxModel();
        //ligar o modelo ao control JComboBox
        cmbAlfombras.setModel(modeloAlfombras);
        txtAlto.setDocument(new LimiteLonxitudeJTextField(5));
        txtAncho.setDocument(new LimiteLonxitudeJTextField(5));
    }
```

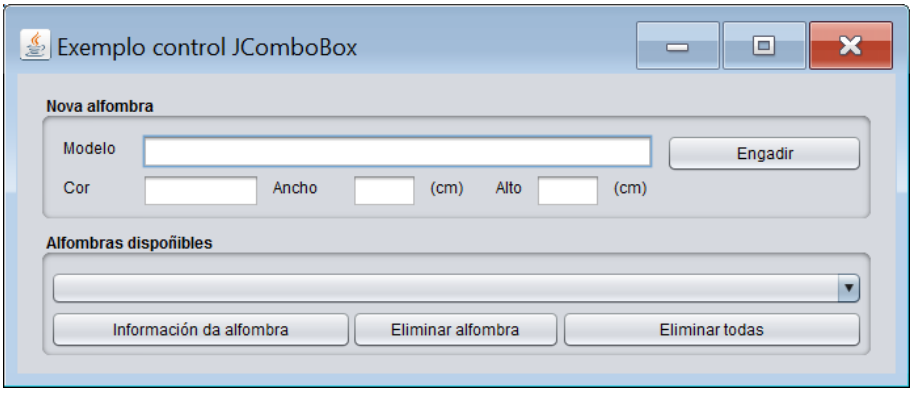
Coa liña:

```
        modeloAlfombras=new DefaultComboBoxModel();
```

reservamos memoria para o almacén de datos (opcionalmente poderíase facer na propia declaración). E con a liña:

```
        cmbAlfombras.setModel(modeloAlfombras);
```

ligamos o almacén de datos ao compoñente gráfico. Unha vez ligado o almacén de datos ao compoñente gráfico, cando executemos a aplicación no compoñente gráfico amosase o contido do almacén de datos, o cal inicialmente está baleiro:



A continuación imos ver como engadimos un elemento ao combo. Cando pulsamos o botón Engadir, valídase o formulario. Se todo é correcto créase un obxecto de tipo Alfombra cos datos introducidos e executamos a seguinte liña:

```
        modeloAlfombras.addElement(alfombra);
```

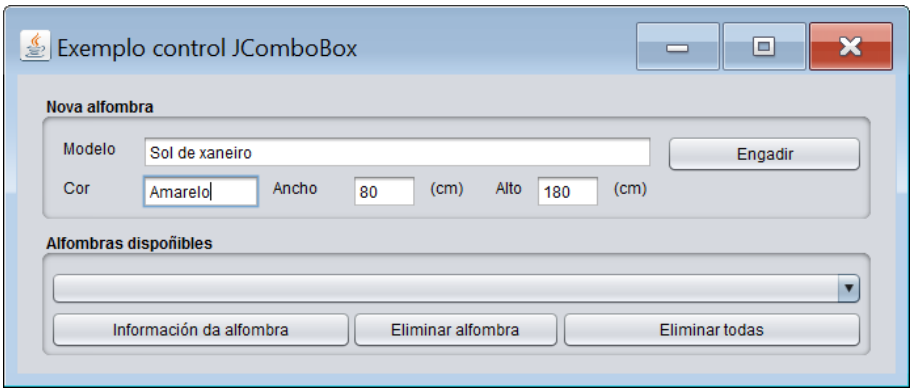
Con esta liña estamos introducindo o obxecto alfombra (de clase Alfombra) na estrutura de datos ligada ao compoñente gráfico. Automaticamente engadirase

o obxecto no compoñente combo, amosándose no compoñente o valor devolto polo método toString do obxecto que acabamos de engadir ao combo (paradigma vista controlador, ver compoñente lista). No noso caso, o método toString da clase Alfombra é o seguinte:

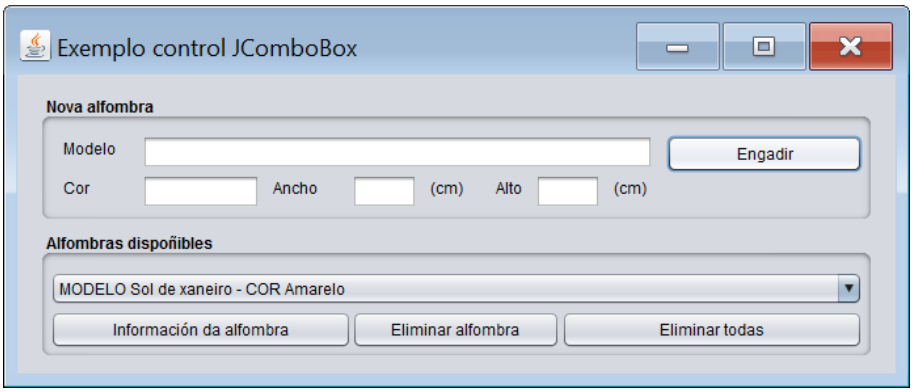
```
@Override
public String toString() {
    return "MODELO " + modelo + " - COR " + cor;
}
```

Neste caso, devolve a cadea "MODELO " concatenada co atributo modelo concatenada coa cadea "- COR: " concatenada co atributo color.

Por exemplo se inserimos os seguintes valores:



Cando pulsemos sobre o botón Engadir este será o resultado:



O almacén interno ligado ao compoñente gráfico garda un obxecto de clase Alfombra cuxos atributos son modelo: Sol de xaneiro, cor: Amarelo, ancho: 80 e alto 180, pero o compoñente gráfico unicamente amosa o valor que devolve a aplicación do método toString do obxecto de clase Alfombra.

A continuación imos ver como implementar o botón Información da alfombra. Ao pulsar sobre este botón amosarase toda a información do obxecto seleccionado no combo. Este é o código asociado ao botón:

```
private void btnInfoAlfombraActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:

    // Comprobar se hai alfombras no combo
    if(modeloAlfombras.getSize()==0)
    {
        JOptionPane.showMessageDialog(this, "Non hai alfombras dispoñibles");
        return;
    }

    //Comprobar se seleccionamos algunha alfombra
```

```
        if(cmbAlfombras.getSelectedIndex()==-1)
        {
            JOptionPane.showMessageDialog(this, "Debe seleccionar ao menos unha alfombra");
            return;
        }

        //Recuperar a alfombra do modelo

        Alfombra alfombra=(Alfombra)modeloAlfombras.getElementAt(cmbAlfombras.getSelectedIndex());
        //Mostras información das alfombra por pantalla
        JOptionPane.showMessageDialog(this, "MODELO: "+alfombra.getModelo()+"\nCOR: "+
                alfombra.getCor()+  "\nANCHO:  "+alfombra.getAncho()+"  cm\nALTO:  "+alfombra.getAlto()+"
cm");

    }
```

O primeiro que temos que facer é comprobar se o combo está baleiro, xa que de ser así, non imos ter información que amosar. Para elo executamos a seguinte liña:

```
        if(modeloAlfombras.getSize()==0)
```

O método getSize aplícase sobre o almacén de datos que é o que realmente conten a información. Devólvenos o número de elementos que conten o almacén de datos.

A continuación comprobamos se o usuario seleccionou algún elemento da lista. Para elo executamos a seguinte liña:

```
        if(cmbAlfombras.getSelectedIndex()==-1)
```

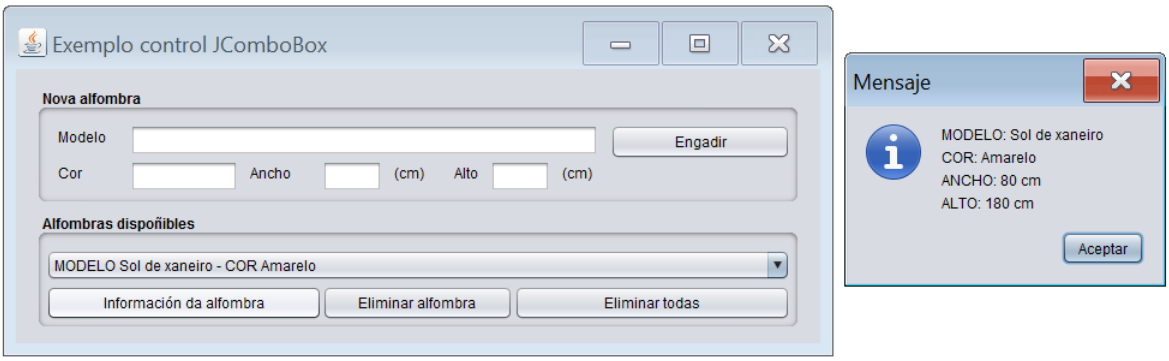
O método getSelectedIndex aplícase sobre o compoñente combo e devolve o índice do elemento seleccionado no combo (o primeiro é o 0). No caso de que devolva -1 quere dicir que non hai ningún elemento marcado.

Finalmente imos recuperar o elemento seleccionado no combo. Para elo empregamos o método getElementAt, o cal aplícase sobre a estrutura de datos que conten a información:

```
        Alfombra alfombra=(Alfombra)modeloAlfombras.getElementAt(cmbAlfombras.getSelectedIndex());
```

Como pódese observar unicamente hai que pasarlle como parámetro a posición do elemento que queremos recuperar a través do método getSelectedIndex. Como no seu momento non tipamos o obxecto modeloAlfombras, o seu método getElementAt vainos devolver un obxecto de clase Object (aínda que realmente o obxecto almacenado é de clase Alfombra). Para poder asignar este obxecto de clase Object a unha variable de clase Alfombra é necesario facerlle un cast. Como pódese observar, o emprego de almacenes de datos xenéricos complica lixeiramente a programación ao forzarnos a levar un control exhaustivo sobre a seu contido, pero por outra banda proporciónanos unha tremenda funcionalidade ao poder almacenar neles calquera tipo de obxecto. Será responsabilidade do programador decidir se é rendible ou non empregar almacenes de datos tipados, tendo en conta que o tipado introduce unha carga extra de funcionalidade no programa e polo tanto reduce o seu rendemento. Unha vez que temos recuperado un obxecto da clase Alfombra podemos acceder aos seus atributos a través dos seus métodos getter e setter.

A continuación amósase o resultado de premer sobre o botón Información da alfombra:



O seguinte botón que imos ver como implementar é o botón Eliminar alfombra. Ao premer sobre este botón elimínase a alfombra seleccionada no combo. Unha vez realizadas as validacións, a liña que se encarga de eliminar o elemento seleccionado no combo é a seguinte:

```
modeloAlfombras.removeElementAt(cmbAlfombras.getSelectedIndex());
```

O método removeElementAt aplícase sobre o almacén de datos. Mediante este método elimínase o obxecto almacenado na posición que se pasa como parámetro no almacén de datos. Debido á ligazón establecida entre o almacén de datos e o compoñente gráfico combo, ao eliminar o obxecto do almacén de datos, tamén desaparece do combo.

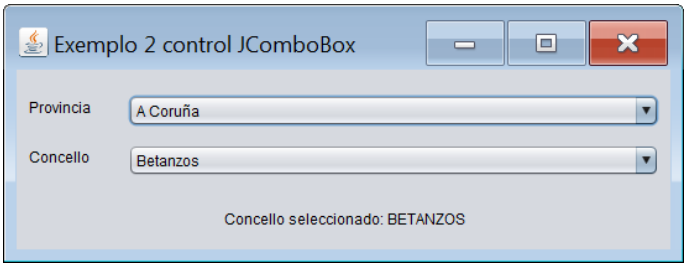
O último botón que imos ver como implementar é o botón Eliminar todas. Ao premer sobre este botón elimínanse tódalas alfombra do combo. Unha vez realizadas as validacións, a liña que se encarga de eliminar tódolos elementos do combo é a seguinte:

```
modeloAlfombras.removeAllElements();
```

O método removeAllElements aplícase sobre o almacén de datos. Mediante este método elimínanse tódolos obxectos almacenados no almacén de datos. Debido á ligazón establecida entre o almacén de datos e o compoñente gráfico combo, ao eliminar tódolos obxectos do almacén de datos, tamén desaparecen do combo.

Xestión de combos empregando NetBeans (segundo exemplo)

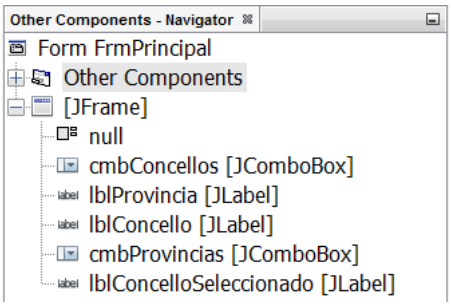
Imos desenvolver outro exemplo para explicar como asignar valores a un combo a través do entorno de programación e como xestionar o seu evento itemStateChanged. Para elo desenvolveremos o seguinte formulario:



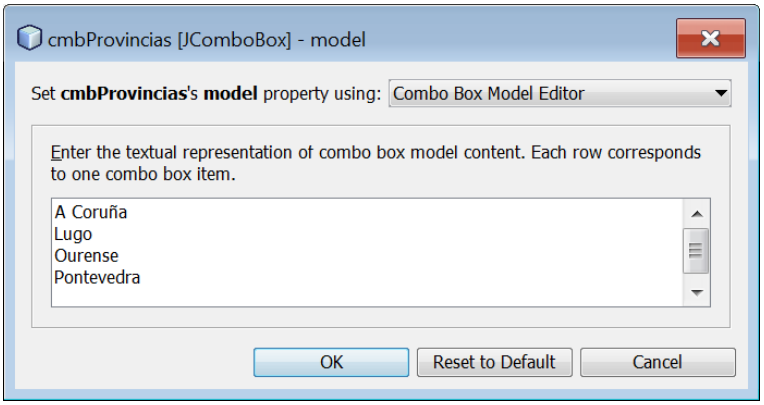
O combo provincia vai ter unha serie de valores establecidos a través do entorno de programación. Cada vez que seleccionemos un valor do combo provincia,

empregando o evento itemStateChanged, cargaranse no combo concello tódolos concellos da provincia seleccionada no primeiro combo. Ao seleccionar un concello no combo concello, empregando o evento itemStateChanged, amosarase nunha etiqueta o nome do concello seleccionado.

O noso formulario quedará configurado do seguinte xeito:



Para asignar os valores que debe ter o noso combo cmbProvincias, picamos sobre o botón asociado á súa propiedade model. Amósasenos unha xanela na cal introducimos os valores que queremos que se amosen no combo:



Temos que ter en conta que se inicializamos un combo deste xeito, os obxectos almacenados serán de tipo String.

O seguinte que queremos é que cada vez que seleccionemos un valor do combo provincia, sexan cargados no combo concello tódolos concellos da provincia seleccionada. A maneira máis eficiente de resolver este problema é mediante o emprego dos eventos asociados ao combo, en concreto o evento itemStateChanged, que é o evento que se activa no momento que cambia o estado dun elemento da lista. Unha pequena aclaración sobre o cambio de estado dun elemento: o cambio de estado pode deberse a dúas razóns. A primeira é que desmarcamos o elemento e a segunda é que o marcamos. Tipicamente cando temos un elemento seleccionado no combo e pasamos a marcar outro execútase o evento dúas veces. A primeira vez porque deseleccionamos o elemento do combo marcado previamente, e unha segunda vez porque marcamos un novo elemento no combo. Unha vez aclarado isto implementamos o código para o evento itemStateChanged do combo cmbProvincia:

```
    private void cmbProvinciasItemStateChanged(java.awt.event.ItemEvent evt) {
        // TODO add your handling code here:
        if(evt.getStateChange()==ItemEvent.SELECTED)
        {
            int posicionProvincia=cmbProvincias.getSelectedIndex();
```

```
            cargarConcellos(posicionProvincia);
        }
```

No caso que estamos tratando, interésanos desenvolver a acción no momento que un novo elemento é seleccionado no combo. Para detectar que un elemento do combo foi seleccionado empregamos a información recibida polo método que implementa o evento a través do seu parámetro evt. O obxecto evt contén información referente ao evento ocorrido. Entre esa información atópase a que indica se o evento foi disparado por unha deselección o por unha nova selección. Para acceder a esta información, facémolo a través do método evt.getStateChange(), o cal devolveranos ItemEvent.SELECTED no caso de que esteamos ante unha nova selección ou devolveranos ItemEvent.DESELECTED no caso de que esteamos ante unha deselección. Unha vez que comprobamos que estamos ante unha nova selección, chamamos ao método cargarConcellos pasándolle a posición da provincia no combo (a fin de simplificar a aplicación non estamos empregando unha base de datos, así que imos supoñer que a posición do elemento no combo é o seu código de rexistro). O método cargarConcellos simplemente cargará os concellos no combo cmbConcellos (como a fin de simplificar a aplicación non estamos empregando unha base de datos, imos ter declarados uns arrais de tipo String con algúns concellos)

O último problema que temos que solucionar é o que se refire a que cando seleccionemos un concello no combo cmbConcellos amósese na etiqueta lblConcelloSeleccionado o nome do concello seleccionado no combo. De novo, o xeito máis eficiente de resolver este problema é facendo emprego do evento itemStateChanged. O código que implementaremos neste caso é o seguinte:

```
    private void cmbConcellosItemStateChanged(java.awt.event.ItemEvent evt) {
        // TODO add your handling code here:
        if(evt.getStateChange()==ItemEvent.SELECTED)
        {
       lblConcelloSeleccionado.setText("Concello seleccionado: "
                   +modeloConcellos.getElementAt(cmbConcellos.getSelectedIndex()).toUpperCase());
        }
    }
```

Simplemente, cada vez que seleccionamos un novo concello no combo cmbConcellos, recupérase o seu nome do almacén de datos asociado ao combo amósase na etiqueta lblConcelloSeleccionado.

Se executamos a aplicación este será o resultado: