Xestión de compoñentes avanzados

Cando desenvolvemos aplicacións gráficas de usuario cada un dos formularios que compoñen as nosas aplicacións están compostos por unha serie de compoñentes gráficos que son empregados para levar a cabo a comunicación de información entre o usuario e a aplicación. Acabamos de ver os compoñentes gráficos básicos. Estes compoñentes unicamente sérvennos para xestionar informacións sinxelas. Cando a información que queremos xestionar ten unha maior complexidade e ademais presenta a característica de estruturarse en coleccións necesitamos outros compoñentes para xestionala. Estes compoñentes son os compoñentes avanzados, os cales estudaremos de seguido. Ao igual que ocorría cos compoñentes gráficos básicos, os compoñentes gráficos avanzados son de emprego moi habitual en calquera aplicación gráfica de usuario. Do mesmo xeito que fixemos cos compoñentes gráficos básicos, centrarémonos nas tarefas para as que se soen empregar estes compoñentes avanzados máis habitualmente, xa que estudar calquera destes compoñentes en profundidade sería un traballo por un lado excesivo, debido a que son moitísimas as propiedades, eventos e métodos que xestiona cada un deles, e por outra banda sería unha perda de tempo, xa que estudaríamos funcionalidades que rara vez vanse empregar. É mellor idea saber xestionar cada compoñente para empregar as súas funcionalidades habituais e no caso de ter que empregar algunha funcionalidade excepcional, aprender como xestionar esa funcionalidade excepcional en concreto. Para coñecer tódalas propiedades, eventos e métodos de cada un dos compoñentes avanzados sempre poderemos acceder á API (application programming interface) de cada compoñente. De seguido imos enumerar os diferentes compoñentes avanzados que imos estudar:

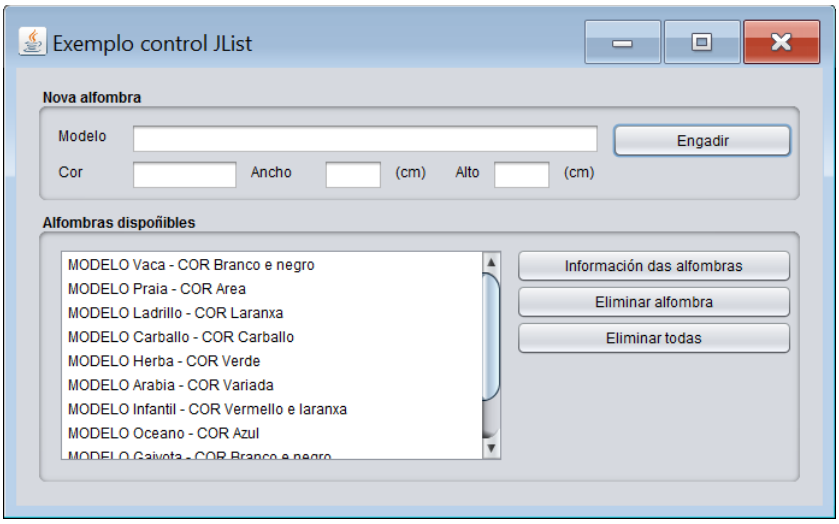
- Lista
- Combo
- Táboa

Compoñente Lista (JList)

Cando nun programa os datos de entrada que solicitamos ao usuario son valores que unicamente poden ter un valor de varios posibles, vimos que o control máis axeitado para recuperar a información é o compoñente de tipo botón de opción, pero as veces o número posible de valores fai inviable o emprego deste control por unha mera razón de espazo (p.e., non ten sentido facer un formulario no cal teñamos que elixir un dos concellos de Galicia a base de botóns de opción. O número de eles que necesitaríamos sería excesivamente elevado). Nestes casos unha das posibles opcións é empregar unha lista que substitúa a tódolos botóns de opción. A lista é un compoñente que contén unha serie de posibles valores dos cales o usuario pode elixir un (segundo a configuración da lista será posible elixir máis dun valor) pero visualmente só amosa algúns dos valores permitindo desprazarnos polos diferentes valores que contén mediante o teclado e o rato. Ademais, as listas presentan outras vantaxes, sendo a máis destacable que a información que contén pode ser modificada dinamicamente en función de distintos eventos. A xestión das listas en java fai emprego do paradigma vista controlador o cal baséase en que o compoñente internamente garda a información que xestiona empregando unha estrutura de datos determinada mentres que o que amosa non ten que coincidir coa información almacenada internamente, senón que é o programador o que pode decidir que é o que se amosara no compoñente gráfico. Basicamente, o modelo vista controlador pódese resumir en que unha cousa é o que hai e outra é o que se ve. O

XUNTA DE GALICIA
Financiado pola Unión Europea NextGenerationEU
Unión Europea Fondo Social Europeo
GOBIERNO DE ESPAÑA MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL
Plan de Recuperación, Transformación y Resiliencia
Xacobeo 21·22

compoñente lista defínese a través da clase javax.swing.JList. Graficamente o seu aspecto é o seguinte:



Na imaxe anterior podemos ver unha lista que contén unha serie de valores. Visualmente non se poden visualizar tódolos valores, non obstante o compoñente permite ao usuario desprazarse polos seus contidos para acceder a tódolos valores posibles que contén.

Propiedades do control JList 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 contén a información que se almacena na lista
SelectedIndex	Índice do elemento seleccionado na lista (o primeiro é o 0). Con -1 indicamos que non hai ningún elemento seleccionado.
SelectionBackground	Cor de fondo do elemento seleccionado
SelectionForeground	Cor do texto do elemento seleccionado
SelectionMode	Tipo de selección que podemos facer sobre os elementos da lista. Pode ser unha selección SINGLE (só pódese seleccionar un elemento da lista), SINGLE_INTERVAL (pódense seleccionar varios elementos da lista pero ten que ser contiguos) ou MULTIPLE_INTERVAL (pódense seleccionar varios elementos da lista e non ten porque ser contiguos)
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

XUNTA DE GALICIA
Financiado pola Unión Europea NextGenerationEU
Unión Europea Fondo Social Europeo
GOBIERNO DE ESPAÑA MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL
Plan de Recuperación, Transformación y Resiliencia
Xacobeo 21·22

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

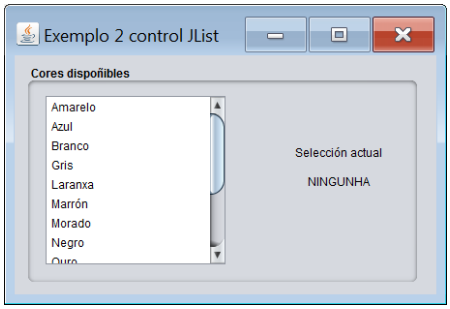
Evento	Lanzamento
MouseClicked	Este evento é disparado cando facemos clic co rato sobre o control. Permite contar o número de clics realizados, polo cal, realmente é habitual empregalo para detectar o dobre clic sobre algún elemento da lista.
ValueChanged	Este evento é disparado cando cambiamos o elemento seleccionado na lista.

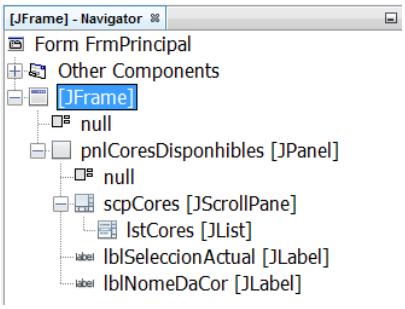
Métodos do control JList empregados máis habitualmente:

Método	Función
public int getSelectedIndex()	Devolve o índice do elemento seleccionado na lista (o primeiro elemento é o 0). Devolve -1 se non hai ningún elemento seleccionado.
public int[] getSelectedIndices()	Devolve os índice dos elemento seleccionados na lista. Devolve un arrai baleiro se non hai ningún elemento seleccionado.
public void setSelectedIndex(int index)	Establece o elemento seleccionado na lista a través do seu índice (o primeiro elemento é o 0).
public void setSelectedIndices(int[] indices)	Establece os elementos seleccionados na lista a través dos seus índices (o primeiro elemento é o 0). As posicións dos elementos marcados pásanse a través dun arrai de int. A lista debe ser multiselección.

Xestión de listas empregando NetBeans

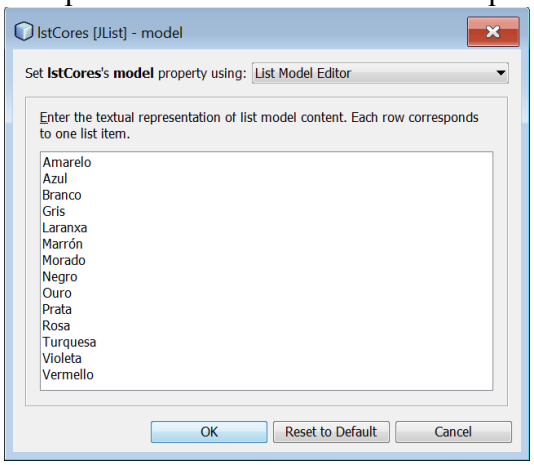
Imos desenvolver un exemplo para explicar como asignar valores á lista a través do entorno de programación e como xestionar o seu evento valueChanged. Para elo desenvolveremos o seguinte formulario:





A lista inicialmente vai ter unha serie de valores establecidos a través do entorno de programación. Cada vez que seleccionemos un valor da lista, empregando o evento valueChanged, amosarase na etiqueta o nome da cor seleccionada.

Para asignar os valores que debe ter a nosa lista, prememos sobre o botón asociado á súa propiedade model. Amósasenos unha xanela na cal podemos introducir os valores que queremos que sexan amosados inicialmente na lista:



Temos que ter en conta que se inicializamos unha lista deste xeito, os obxectos almacenados na lista serán de tipo String.

Respecto á segunda parte do programa, é dicir, a parte na que cada vez que seleccionemos un valor da lista amósase na etiqueta o nome da cor seleccionada, o xeito máis eficiente de resolvela é mediante o emprego dos eventos asociados á lista, en concreto o evento **valueChanged**, que é o evento que se activa no momento que cambia o elemento seleccionado na lista. Para quen poida preguntarse porqué non empregamos o evento mouseClicked, dicir que a razón é que tamén podemos cambiar o elemento seleccionado na lista co teclado e mouseClicked non é capaz de detectalo. Isto obrigaríanos a implementar tamén un evento de teclado sobre a lista. Non obstante, o evento valueChanged detecta cambios na selección da lista sin importarlle como foi producido, xa fora por rato ou por teclado. Unha vez aclarado isto implementamos o código para o evento valueChanged da lista:

```
    private void lstCoresValueChanged(javax.swing.event.ListSelectionEvent evt) {
        // TODO add your handling code here:
        if(!evt.getValueIsAdjusting())
        {
            String corSeleccionada=(String)lstCores.getModel().getElementAt(lstCores.getSelectedIndex());
            if(corSeleccionada.trim().compareTo("")==0)
            {
                lblNomeDaCor.setText("NINGUNHA");
            }
            else
            {
                lblNomeDaCor.setText(corSeleccionada.toUpperCase());
            }
        }
    }
```

Respecto a este evento hai que mencionar algo moi importante: se é disparado por unha acción de teclado execútase unha vez, pero se é disparado debido á acción do rato execútase dúas ou máis veces. A razón de que co rato execútese dous ou máis veces é que cando prememos o rato e mentres non o soltamos podemos seleccionar un elemento da lista movendo o punteiro a través da lista. Cada vez que pasamos por encima dun elemento da lista, considérase unha nova elección e polo tanto dispárase o evento. Ademais no momento que soltemos o botón do rato vólvese a disparar o evento xa que considera que esa é outra nova selección. Non obstante co teclado, ao premer a tecla arriba ou a tecla abaixo unicamente podemos seleccionar o anterior ou o seguinte elemento, polo cal só hai unha selección e polo tanto o evento é lanzado unha única vez. Para solucionar o problema de múltiples disparos do evento valueChanged debido ao emprego do rato facemos uso do método getValueIsAdjusting, o cal devolverá false no momento no cal soltemos o rato sobre a selección

final. Como pódese observar é o que facemos no código implementado para o evento valueChanged:

```
        if(!evt.getValueIsAdjusting())
```

Respecto á recuperación do elemento seleccionado, cabe a pena mencionar que á lista non lle asignamos ningún almacén de datos. Para acceder ao almacén de datos preestablecido polo entorno de programación empregamos o método getModel sobre o compoñente lista. O método getModel devolveranos o almacén de datos asociado á lista. Unha vez conseguido o almacén de datos, podemos empregar os métodos para acceder aos datos almacenados no almacén de datos:

```
String corSeleccionada=(String)lstCores.getModel().getElementAt(lstCores.getSelectedIndex());
```

Se executamos a aplicación este será o resultado se seleccionamos unha cor.

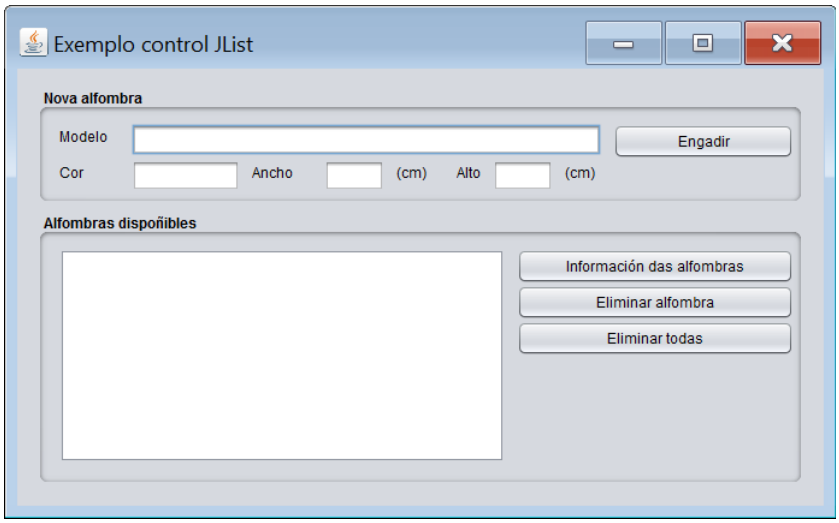
A continuación amósase o listado do código fonte desenvolvido para a realización da aplicación

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

import java.awt.event.KeyEvent;
import javax.swing.DefaultListModel;
import javax.swing.JOptionPane;

public class FrmPrincipal extends javax.swing.JFrame {

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


    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        pnlCoresDisponhibles = new javax.swing.JPanel();
        scpCores = new javax.swing.JScrollPane();
        lstCores = new javax.swing.JList();
        lblSeleccionActual = new javax.swing.JLabel();
        lblNomeDaCor = new javax.swing.JLabel();
e().setLayout(null);

        pnlCoresDisponhibles.setBorder(javax.swing.BorderFactory.createTitledBorder("Cores dispoñibles"));
        pnlCoresDisponhibles.setLayout(null);

        lstCores.setModel(new javax.swing.AbstractListModel() {
             String[] strings = { "Amarelo", "Azul", "Branco", "Gris", "Laranxa", "Marrón", "Morado", "Negro", "Ouro", "Prata",
"Rosa", "Turquesa", "Violeta", "Vermello" };
            public int getSize() { return strings.length; }
            public Object getElementAt(int i) { return strings[i]; }
        });
        lstCores.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
        lstCores.addListSelectionListener(new javax.swing.event.ListSelectionListener() {
            public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
                lstCoresValueChanged(evt);
            }
        });
        scpCores.setViewportView(lstCores);

        pnlCoresDisponhibles.add(scpCores);
        scpCores.setBounds(20, 30, 190, 178);

        lblSeleccionActual.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        lblSeleccionActual.setText("Selección actual");
        pnlCoresDisponhibles.add(lblSeleccionActual);
        lblSeleccionActual.setBounds(260, 80, 130, 20);

        lblNomeDaCor.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        lblNomeDaCor.setText("NINGUNHA");
        pnlCoresDisponhibles.add(lblNomeDaCor);
        lblNomeDaCor.setBounds(260, 110, 130, 20);

        getContentPane().add(pnlCoresDisponhibles);
        pnlCoresDisponhibles.setBounds(10, 10, 410, 230);

        java.awt.Dimension screenSize = java.awt.Toolkit.getDefaultToolkit().getScreenSize();
```

```
        setBounds((screenSize.width-458)/2, (screenSize.height-311)/2, 458, 311);
    }// </editor-fold>

    private void lstCoresValueChanged(javax.swing.event.ListSelectionEvent evt) {
        // TODO add your handling code here:
        if(!evt.getValueIsAdjusting())
        {
            String corSeleccionada=(String)lstCores.getModel().getElementAt(lstCores.getSelectedIndex());
            if(corSeleccionada.trim().compareTo("")==0)
            {
                lblNomeDaCor.setText("NINGUNHA");
            }
            else
            {
                lblNomeDaCor.setText(corSeleccionada.toUpperCase());
            }
        }
    }

    private boolean isNumero(java.awt.event.KeyEvent evt)
    {
        // Comprobar se a pulsacion de teclado pasada
        // en evt é un número
        char caracter=evt.getKeyChar();
        if((
            (caracter<'0' || caracter >'9')&&
            (caracter!=KeyEvent.VK_SPACE) &&
            (caracter!=KeyEvent.VK_BACK_SPACE)
            ) ||
            (evt.isAltDown() ||
             evt.isAltGraphDown() ||
             evt.isControlDown() ||
             evt.isMetaDown()
           ))
        {
            return false;
        }
        return true;
    }

    /**
     * @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.JLabel lblNomeDaCor;
    private javax.swing.JLabel lblSeleccionActual;
    private javax.swing.JList lstCores;
    private javax.swing.JPanel pnlCoresDisponhibles;
    private javax.swing.JScrollPane scpCores;
    // End of variables declaration
}
        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Exemplo 2 control JList");
        getContentPan
```

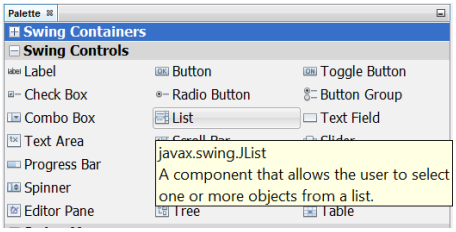
Xestión de listas empregando NetBeans

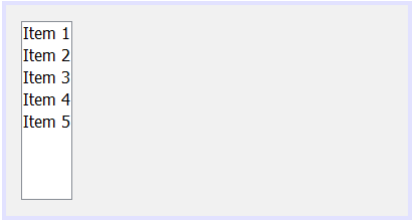
A continuación imos desenvolver o seguinte formulario:



O formulario consta dunha lista, a cal atópase no panel de alfombras dispoñibles. Será empregado para engadir nela as alfombras que definamos desde o panel nova alfombra. A través dos tres botóns adxuntos á lista poderemos realizar diferentes accións sobre os elementos da lista. Ademais, no caso de que fagamos dobre clic sobre algún elemento da lista, amosarase a información ao respecto dese elemento. A lista será de tipo multiselección de intervalo múltiple. 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 JList no noso formulario primeiramente seleccionarémolo da paleta de compoñentes do entorno de programación e arrastrámolo ata o formulario:

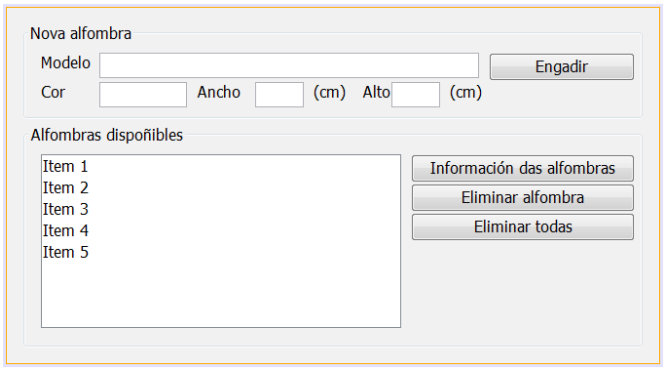


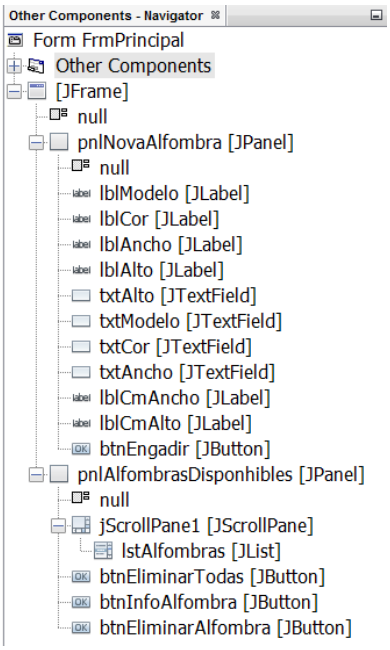


Cando arrastramos unha lista ao noso formulario en realidade créanse dous compoñentes: un JScrollPane e dentro deste a lista. Isto é debido a que cando engadimos elementos dentro do compoñente lista, pode ocorrer que haia máis elementos na lista dos que se poden visualizar nela. Para desprazarnos polos elementos da lista podemos empregar os cursores, aínda que é máis fácil facelo mediante barras de desprazamento. O problema é que o compoñente lista non ten barras de desprazamento. Aquí é onde entra en acción o entorno de programación creando a lista dentro dun JScrollPane. Como resultado, temos que aínda que a lista non ten barras de desprazamento, de cara ao usuario aparenta que as ten xa que estas son provistas polo JScrollPane.

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 unha lista. Unha vez engadidos e colocados no seu lugar correspondente, este será o resultado do deseño:



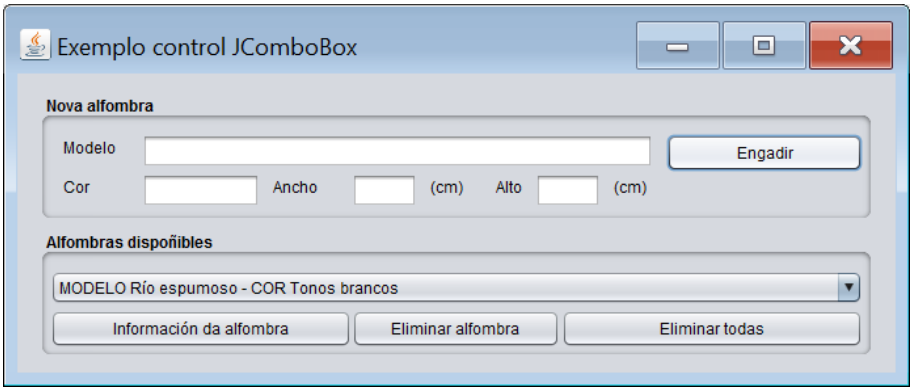


Xestión de combos empregando NetBeans (exemplo)

1.1.1.1 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:



XUNTA DE GALICIA
Financiado pola Unión Europea NextGenerationEU
Unión Europea Fondo Social Europeo
GOBIERNO DE ESPAÑA MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL
Plan de Recuperación, Transformación y Resiliencia
Xacobeo 21·22

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:

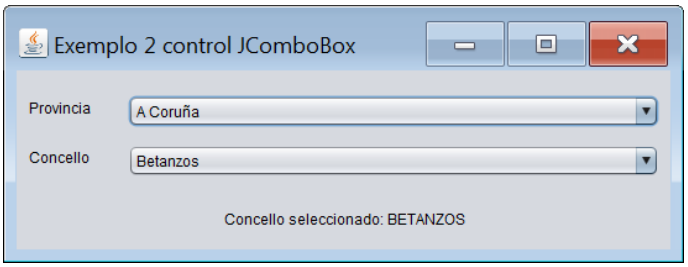
Evento	Lanzamento
ItemStateChanged	Este evento é disparado cando cambiamos o estado dun elemento do combo. Este evento pode dispararse en dúas situacións: a primeira situación dáse cando deseleccionamos un elemento e a segunda situación dase cando seleccionamos un elemento. O evento prové información 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.

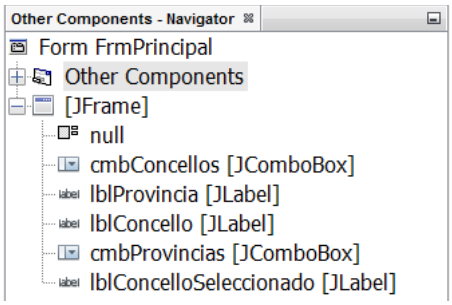
Xestión de combos empregando NetBeans (exemplo)

Imos desenvolver un 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:

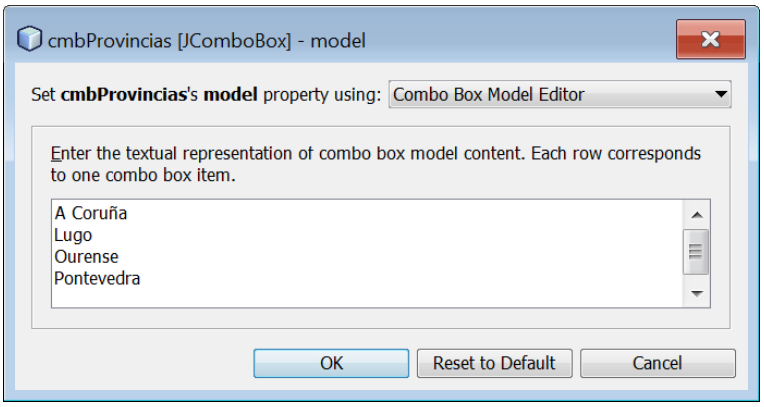


O combo provincia vai ter unha serie de valores establecidos a través do entorno de programación. Cada vez que seleccionemos un valor do combo provincia, empregando o evento itemStateChanged, cargaranse no combo concello tódolos concellos da provincia seleccionada no primeiro combo. Ao seleccionar un 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);
    }
```

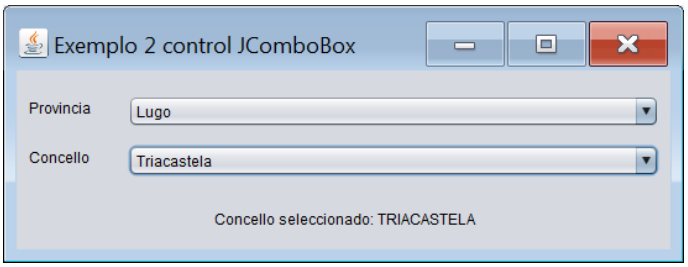
No caso que estamos tratando, interésanos desenvolver a acción no momento que un novo elemento é seleccionado no combo. Para detectar que un elemento do combo foi seleccionado empregamos a información recibida polo método que implementa o evento a través do seu parámetro evt. O obxecto evt contén información 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:



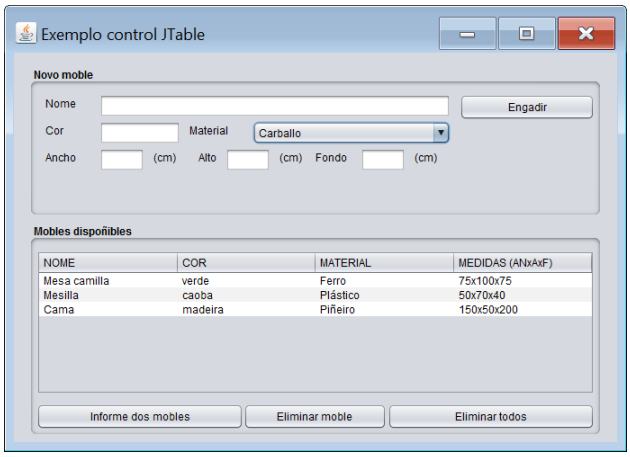
A continuación amósase o listado do código fonte desenvolvido para a realización da aplicación:

```
package exemploJComboBox;

import java.awt.event.ItemEvent;
import java.awt.event.KeyEvent;
import javax.swing.DefaultComboBoxModel;
import javax.swing.DefaultListModel;
import javax.swing.JOptionPane;

public class FrmPrincipal extends javax.swing.JFrame {

    /**
     * Creates new form FrmPrincipal
     */
    public FrmPrincipal() {
        initComponents();
        //crear o modelo para almacenar as alfombras
        modeloConcellos=new DefaultComboBoxModel<String>();
        //ligar o modelo ao control JComboBox
        cmbConcellos.setModel(modeloConcellos);
        cargarConcellos(0);
    }

    /**
     * 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() {

        cmbConcellos = new javax.swing.JComboBox();
        lblProvincia = new javax.swing.JLabel();
        lblConcello = new javax.swing.JLabel();
        cmbProvincias = new javax.swing.JComboBox();
        lblConcelloSeleccionado = new javax.swing.JLabel();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Exemplo 2 control JComboBox");
        getContentPane().setLayout(null);

        cmbConcellos.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Item 1", "Item 2", "Item 3", "Item 4" }));
        cmbConcellos.addItemListener(new java.awt.event.ItemListener() {
            public void itemStateChanged(java.awt.event.ItemEvent evt) {
                cmbConcellosItemStateChanged(evt);
            }
        });
        getContentPane().add(cmbConcellos);
        cmbConcellos.setBounds(90, 60, 430, 26);

        lblProvincia.setText("Provincia");
        getContentPane().add(lblProvincia);
        lblProvincia.setBounds(10, 20, 90, 20);

        lblConcello.setText("Concello");
        getContentPane().add(lblConcello);
        lblConcello.setBounds(10, 60, 60, 20);

        cmbProvincias.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "A Coruña", "Lugo", "Ourense", "Pontevedra" }));
        cmbProvincias.addItemListener(new java.awt.event.ItemListener() {
            public void itemStateChanged(java.awt.event.ItemEvent evt) {
                cmbProvinciasItemStateChanged(evt);
            }
        });
        getContentPane().add(cmbProvincias);
        cmbProvincias.setBounds(90, 20, 430, 26);

        lblConcelloSeleccionado.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        lblConcelloSeleccionado.setText("Concello seleccionado: -");
        getContentPane().add(lblConcelloSeleccionado);
        lblConcelloSeleccionado.setBounds(10, 110, 510, 20);

        java.awt.Dimension screenSize = java.awt.Toolkit.getDefaultToolkit().getScreenSize();
        setBounds((screenSize.width-551)/2, (screenSize.height-209)/2, 551, 209);
    }// </editor-fold>
```

```
    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);
        }

    }

    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());
        }
    }

    private void cargarConcellos(int codProvincia)
    {
        modeloConcellos.removeAllElements();
        switch(codProvincia)
        {
            case 0: for(int i=0;i<concellosCorunha.length;i++)
                    {
                        modeloConcellos.addElement(concellosCorunha[i]);
                    }
                    break;
            case 1: for(int i=0;i<concellosLugo.length;i++)
                    {
                        modeloConcellos.addElement(concellosLugo[i]);
                    }
                    break;
            case 2: for(int i=0;i<concellosOurense.length;i++)
                    {
                        modeloConcellos.addElement(concellosOurense[i]);
                    }
                    break;
            case 3: for(int i=0;i<concellosPontevedra.length;i++)
                    {
                        modeloConcellos.addElement(concellosPontevedra[i]);
                    }
                    break;
        }
    }


    /**
     * @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.JComboBox cmbConcellos;
    private javax.swing.JComboBox cmbProvincias;
    private javax.swing.JLabel lblConcello;
    private javax.swing.JLabel lblConcelloSeleccionado;
    private javax.swing.JLabel lblProvincia;
    // End of variables declaration
    private javax.swing.DefaultComboBoxModel<String> modeloConcellos;

    //Pequena "base de datos" de concellos por provincias
    private String[] concellosCorunha={"Betanzos","Ferrol","Pontedeume"};
    private String[] concellosLugo={"Foz","Quiroga","Triacastela"};
    private String[] concellosOurense={"Bande","Castro Caldelas","Maside"};
    private String[] concellosPontevedra={"Bueu","Marín","Tomiño"};
}
```

Compoñente Táboa (Jtable)

Unha táboa é un compoñente gráfico composto por filas e columnas de celas, sendo nestas últimas nas que se almacena a información que conten a táboa. O compoñente gráfico JTable permite a visualización dunha gran cantidade de información ben organizada dun só golpe de vista. A información contida nun compoñente da clase JTable pode ser modificada dinamicamente en función de distintos eventos. A xestión das táboas en java fai emprego do paradigma vista controlador (ver listas). O compoñente lista defínese a través da clase javax.swing.JTable. Graficamente o seu aspecto é o eguinte:



Na imaxe anterior podemos ver unha táboa que conten unha serie de valores definidos dinamicamente a través do emprego dun programa.

Propiedades do control JTable 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
GridColor	Cor da reixa da táboa
Model	Modelo de datos. Estrutura de datos que contén a información que se almacena na táboa
RowHeight	Altura das filas da táboa

RowSelectionAllowed	Activando esta propiedade indicamos que ao seleccionar un elemento na táboa seleccionarase a fila enteira á que pertence ese elemento. Se desactivamos esta propiedade, ao seleccionar un elemento na táboa unicamente seleccionarase a cela dese elemento.
SelectionBackground	Cor de fondo da selección
SelectionForeground	Cor do texto da selección
SelectionModel	Tipo de selección que podemos facer sobre os elementos da lista. Pode ser unha selección Single Selection (unicamente pódese seleccionar un elemento da táboa), Single Interval Selection (pódense seleccionar varios elementos da táboa, pero deben ser contiguos) ou Multiple Interval Selection (pódense seleccionar varios elementos da táboa e non é preciso que sexan contiguos)
ShowHorizontalLines	Mediante esta propiedade indícase se as liñas horizontais da reixa da táboa serán ou non serán visibles
ShowVerticalLines	Mediante esta propiedade indícase se as liñas verticais da reixa da táboa serán ou non serán visibles
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 JTable empregados máis habitualmente e que pódense establecer a través do entorno de programación:

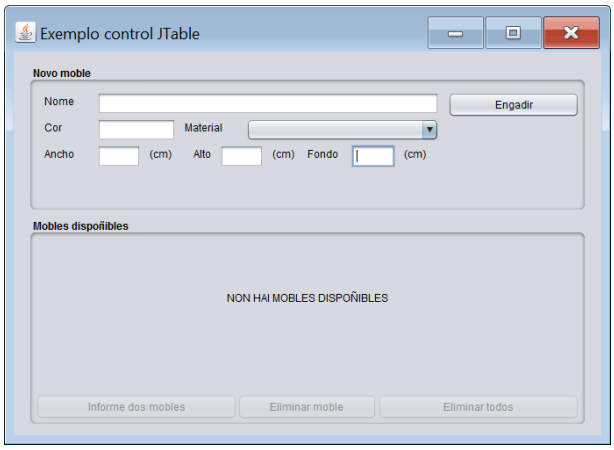
Evento	Lanzamento
MouseClicked	Este evento é disparado cando facemos clic co rato sobre o control. Permite contar o número de clics realizados, polo cal realmente sóese empregar para detectar o dobre clic sobre algún elemento da táboa.

Métodos do control JTable empregados máis habitualmente:

Método	Función
public int getSelectedRow()	Devolve o índice da fila seleccionada na táboa. Devolve -1 se non hai ningunha fila seleccionada.
public int getSelectedRowCount()	Devolve o número de filas seleccionadas
public int[] getSelectedRows()	Devolve os índices das filas seleccionadas na táboa. Devolve un arrai baleiro se non hai ningunha fila seleccionada.

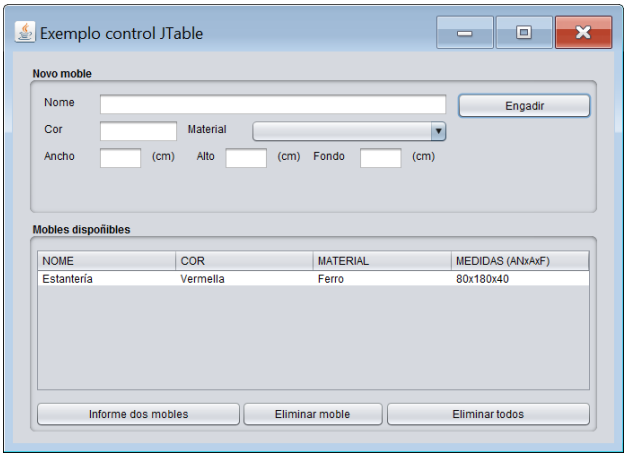
Xestión de táboas empregando NetBeans

A continuación imos desenvolver o seguinte formulario:



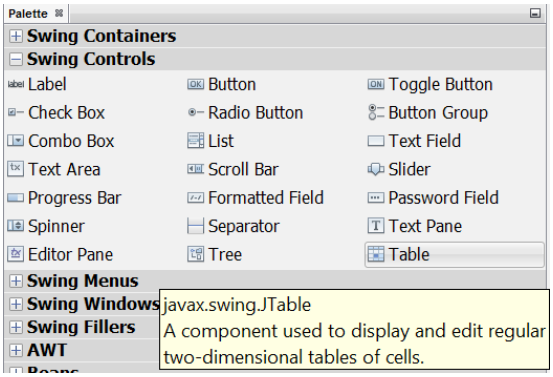
XUNTA DE GALICIA
Financiado pola Unión Europea NextGenerationEU
Unión Europea Fondo Social Europeo
GOBIERNO DE ESPAÑA MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL
Plan de Recuperación, Transformación y Resiliencia
Xacobeo 21·22

O formulario consta dunha táboa, a cal atópase no panel de mobles dispoñibles. A táboa emprégase para engadir nela os mobles que definamos desde o panel novo moble. Como pódese observar na imaxe anterior, a táboa inicialmente non se visualiza. Imos facer que o noso programa só amose a táboa se conten datos. No caso contrario unicamente amosará unha etiqueta informando de que non hai mobles dispoñibles. No caso de que teñamos introducido algún dato na táboa, este será o aspecto da aplicación:

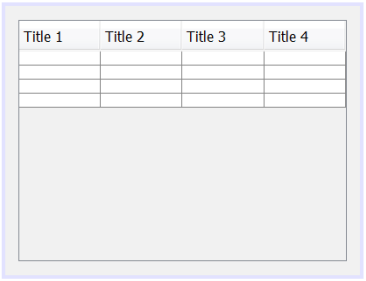


Como pódese observar na imaxe, agora amósase unha táboa, a cal contén información organizada en filas e columnas. Ademais, como un detalle menor fixémonos nos tres botóns que acompañan á táboa. Empréganse para xestionar a información da táboa. Só estarán habilitados cando teñamos información que manipular na táboa. Adicionalmente, no caso de que fagamos dobre clic sobre algunha fila da lista, amosarase a información ao respecto desa fila. A táboa será de tipo multiselección de intervalo múltiple. As caixas de texto Ancho, Alto e Fondo só admitirán números e a súa lonxitude estará limitada a cinco caracteres. Calquera error que poida acontecer ao longo do emprego do programa debe ser reportado ao usuario.

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



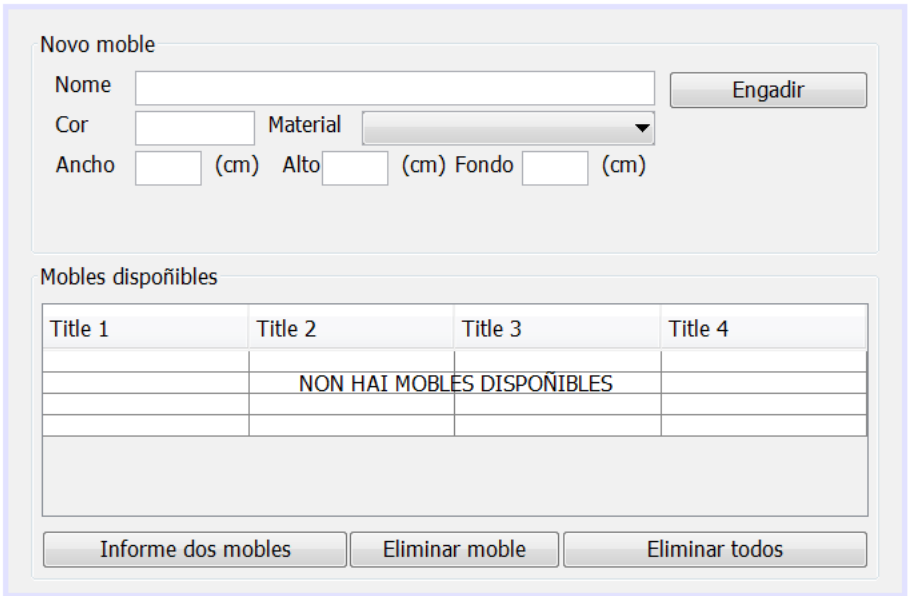
e arrastrámolo ata o formulario:



Cando arrastramos unha táboa ao noso formulario en realidade créanse dous compoñentes: un JScrollPane e dentro deste a táboa. Isto é debido a que cando engadimos elementos dentro do compoñente táboa, pode ocorrer que teñamos máis elementos na táboa dos que se poden visualizar en ela. Para desprazarnos polos elementos da táboa podemos empregar os cursores, aínda que é máis doado facelo mediante barras de desprazamento. O problema é que o compoñente táboa non ten barras de desprazamento. Aquí é onde entra en acción o entorno de programación creando a táboa dentro dun JScrollPane. Como resultado, temos que aínda que a táboa non ten barras de desprazamento, de cara ao usuario aparenta que as ten, xa que estas son provistas polo JScrollPane.

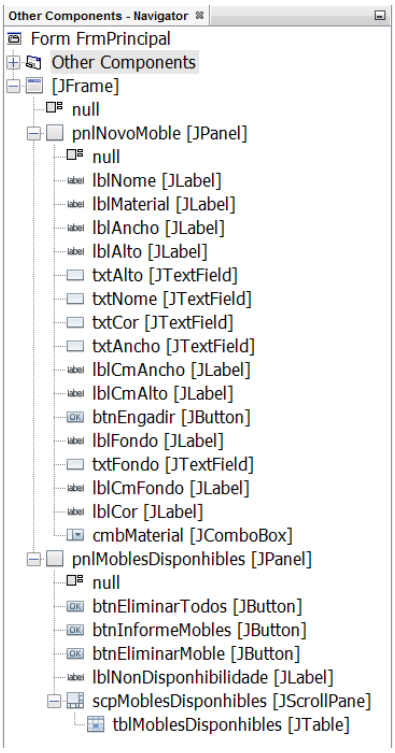
Unha vez situado o compoñente dentro do formulario, adaptámolo para que teña o aspecto visual que desexamos que teña.

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



XUNTA DE GALICIA
Financiado pola Unión Europea NextGenerationEU
Unión Europea Fondo Social Europeo O FSE invirte no teu futuro
GOBIERNO DE ESPAÑA MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL
Plan de Recuperación, Transformación y Resiliencia
Xacobeo 21·22

O noso formulario quedará configurado do seguinte xeito:



Unha vez que temos colocados os compoñentes que necesitamos hai que configurar o seu aspecto. Neste caso faremos as seguintes modificacións:

- Para o combo cmbMaterial, modificamos a súa propiedade model para que teña os seguintes valores: Carballo, Cerdeira, Faia, Ferro, Piñeiro, Plástico. Ademais á súa propiedade selectedIndex asignámoslle o valor -1 para que de entrada ningún valor do combo sexa preseleccionado.
- A tódolos botóns do panel pnlMoblesDisponhibles deshabilitámoslles a súa propiedade enabled para que ao arrancar a aplicación aparezan deshabilitados.

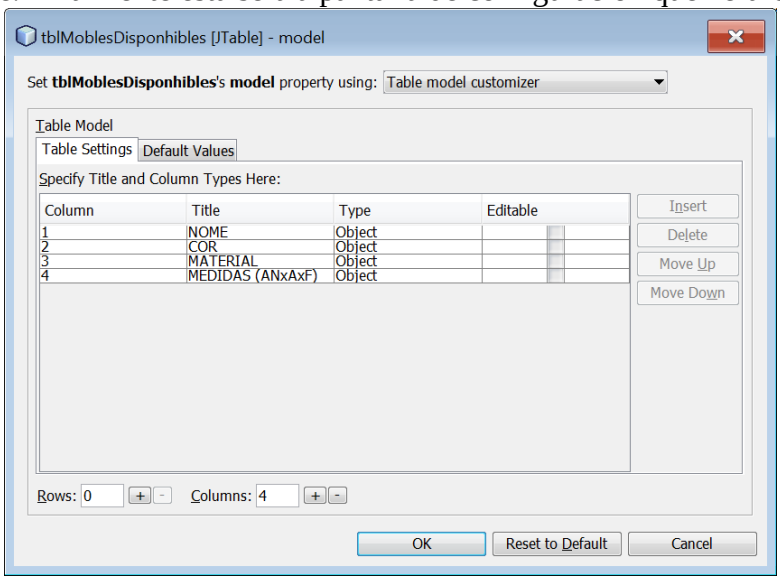
Para facer que a táboa non sexa visible ao iniciar a aplicación, temos que modificar a súa propiedade visible, pero o entorno empregado para o desenvolvemento da aplicación non permite facelo graficamente, así que o temos que facer por código:

```
public FrmPrincipal() {
    initComponents();
    scpMoblesDisponhibles.setVisible(false);
```

O mellor sitio para facer isto é no construtor do formulario xusto despois de pintar os seus compoñentes (initComponents). O método empregado é o método setVisible sobre o compoñente scpMoblesDisponhibles, ao cal pasámoslle false para indicar que non o queremos visualizar. A partir de agora, o compoñente existirá no noso contedor, pero non será visible. En xeral a tódolos compoñentes gráficos pódeselles aplicar este método e serve para facelos visibles (pasando true ao método) o invisibles (pasando false ao método).

Atributos da táboa: imos indicar o número de filas e de columnas da táboa, as cabeceiras da táboa, o tipo de obxectos almacenables na táboa e a posibilidade de edición da táboa. Para elo imos empregar unha pantalla de configuración á que se accede premendo sobre o botón asociado á propiedade model do compoñente tblMoblesDisponhibles. Imos configurar os datos desa pantalla de xeito que obtemos a seguinte configuración para a nosa táboa: cero filas e 4 columnas. Para a

primeira columna, a súa cabeceira será NOME, poderase almacenar nela calquera tipo de obxecto (Object) e non será editable (casilla editable desmarcada). Para a segunda, terceira e carta columna só variarán os textos de as súas cabeceiras, que serán COR, MATERIAL e MEDIDAS (ANxAxF) respectivamente. Finalmente esta será a pantalla de configuración que xeraremos para a nosa táboa:



O comportamento visual da aplicación xa está resolto (unicamente deberemos pulir o detalle de cando amosar ou non a táboa e habilitar ou deshabilitar o seus botóns adxuntos). Agora imos desenvolver a súa funcionalidade.

Anteriormente, mencionamos que o funcionamento da táboa baséase no paradigma vista controlador segundo o cal o compoñente internamente garda a información que xestiona empregando unha estrutura de datos determinada mentres que o que amosa non ten que coincidir coa información gardada internamente, senón que é o programador o que pode decidir que é o amosado no compoñente gráfico. Segundo o que acabamos de explicar, para empregar o paradigma vista controlador necesitamos dous elementos, o compoñente gráfico (tblMoblesDisponhibles) que xa o temos e unha estrutura de datos onde almacenar a súa información. Este almacén de datos será un obxecto da clase javax.swing.table.DefaultTableModel. Este obxecto 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 podamos acceder a el desde calquera método do formulario:

```
    private DefaultTableModel modeloMobles;
```

Unha vez definida a estrutura de datos que vai funcionar como almacén dos datos da táboa, temos que ligala ao compoñente gráfico de tipo táboa que vai amosar os seus datos. Neste caso imos empregar a configuración realizada sobre a propiedade model da táboa e imos indicar que o almacén de datos imos empregar vai ser o definido a través da propiedade model da táboa. Este almacén de datos xa ten predefinidas unha serie de características (cero filas, catro columnas, cabeceiras, tipos de obxectos almacenables e indicación da non posibilidade de editar as celas). Todas estas características tamén poderíanse indicar a través dun construtor específico da clase DefaultTableModel, pero xa que o entorno de programación nos proporciona unha ferramenta para facelo, podemos facer emprego dela.

Resumindo, á variable que imos empregar para referenciar a información almacenada na táboa, ímoslle asignar o modelo definido graficamente para a táboa mediante a seguinte liña de código:

```
        modeloMobles=(DefaultTableModel)tblMoblesDisponhibles.getModel();
```

No construtor do formulario e unha vez pintados os seus compoñentes (método initComponentes), facemos emprego do método getModel sobre a táboa. O método getModel obtén unha referencia ao modelo asignado a un compoñente gráfico. Por ultimo asignamos esta referencia á variable modeloMobles, a cal imos empregar para xestionar o almacén de datos asignado á táboa.

De seguido imos ver como engadimos un elemento á táboa. Cando pulsamos o botón Engadir, valídase o formulario. Se todo é correcto xérase un obxecto de tipo Moble cos datos introducidos, amosamos a táboa e habilitamos os botóns asociados á táboa. Por último executamos as seguintes liñas:

```
//Engadir o moble á táboa
modeloMobles.setRowCount(modeloMobles.getRowCount()+1);
modeloMobles.setValueAt(moble, modeloMobles.getRowCount()-1 , 0);
modeloMobles.setValueAt(moble.getCor(), modeloMobles.getRowCount()-1 , 1);
modeloMobles.setValueAt(moble.getMaterial(), modeloMobles.getRowCount()-1 , 2);
String medidas=moble.getAncho()+"x"+moble.getAlto()+"x"+moble.getFondo();
modeloMobles.setValueAt(medidas, modeloMobles.getRowCount()-1 , 3);
```

Analicemos estas liñas: a primeira liña é a seguinte:

```
modeloMobles.setRowCount(modeloMobles.getRowCount()+1);
```

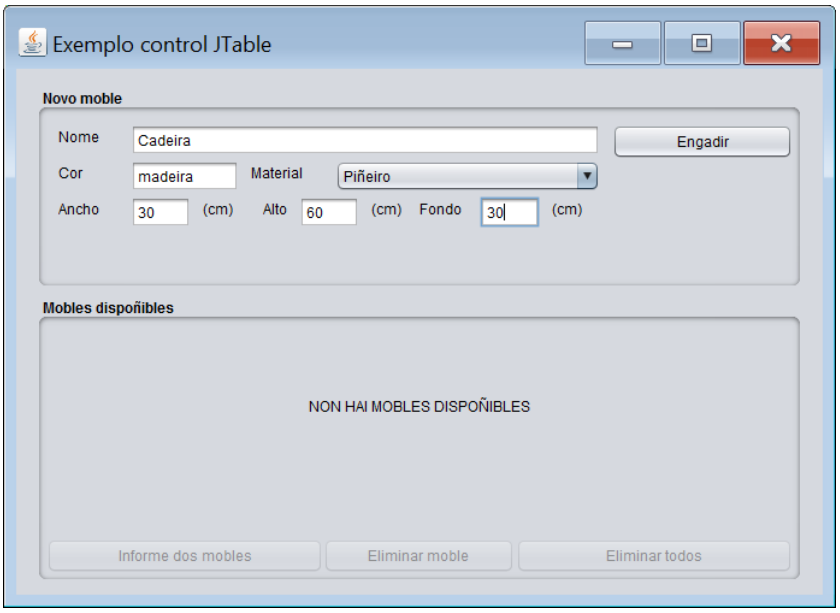
Con esta liña estamos creando unha nova fila ao final do almacén de datos asociado á táboa. Nesta nova fila almacenaremos os obxectos que despois reflectiranse na táboa ligada ao almacén de datos. O método setRowCount establece o número de filas que ten o almacén de datos. O método getRowCount devolve o número de filas que ten o almacén de datos. Neste caso estamos establecendo no almacén de datos tantas filas como ten máis unha, é dicir, estamos engadindo unha fila ao almacén de datos.

As seguintes liñas son as que se empregan para inserir os obxectos na nova fila creada no almacén de datos. Coa seguinte liña de código:

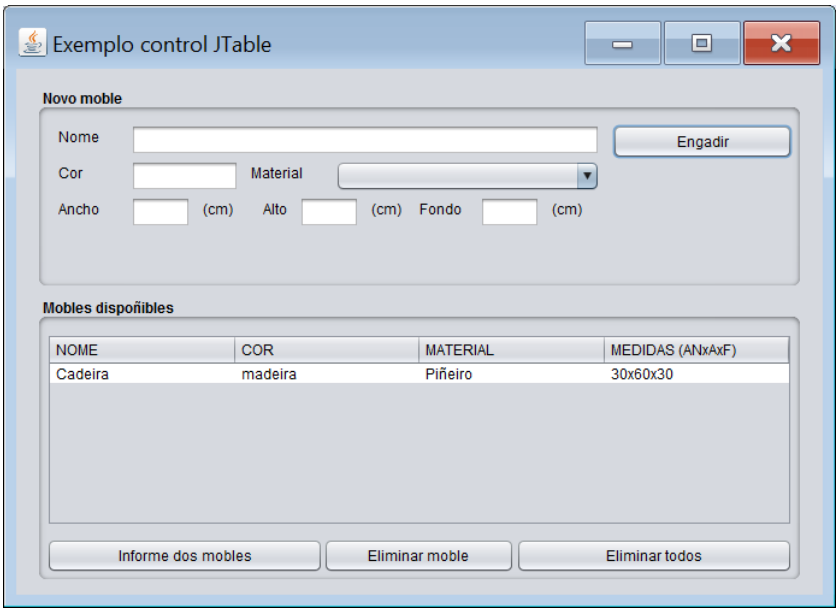
```
modeloMobles.setValueAt(moble, modeloMobles.getRowCount()-1 , 0);
```

Engadimos ao almacén de datos o obxecto moble que conten os datos do novo moble creado. Como o almacén de datos é unha matriz de celas, debemos indicar a fila e a columna donde imos colocar o obxecto. Neste caso, a fila indicámoslla con modeloMobles.getRowCount()-1. Restámoslle un xa que a primeira fila do almacén de datos é a cero. A columna, neste caso é a primeira, a cal no almacén de datos é a cero. Ao igual que ocorría con listas e combos, ao engadir un obxecto no almacén de datos ligado a un compoñente táboa, o obxecto almacénase completo no almacén de datos, pero na cela correspondente do compoñente táboa vaise visualizar unicamente a información que devolve o método toString do obxecto gardado no almacén de datos.

De igual xeito coas seguintes liñas engadimos na nova fila creada no almacén de datos os valores para a segunda, terceira e carta columna, os cales visualizaranse no compoñente táboa. En cada cela podemos almacenar o obxecto que queiramos. Neste caso optouse por almacenar un Moble (que conten toda a información referente a ese moble) na primeira columna do modelo, mentres que nas outras tres almacenáronse obxectos de tipo String.
Por exemplo se inserimos os seguintes valores:



Cando pulsemos sobre o botón Engadir este será o resultado:



O almacén interno ligado ao compoñente gráfico almacena na súa primeira columna un obxecto de clase Moble cuxos atributos son nome: Cadeira, cor: madeira, ancho: 30, alto: 60 e fondo: 30, pero o compoñente gráfico unicamente amosa na primeira columna o valor que devolve a aplicación do método toString do obxecto de clase Moble, é dicir, Cadeira. Respecto á segunda columna do almacén de datos, almacena un String cuxo valor é madeira. Como o método toString dun String é o propio String, na segunda cela do compoñente visual amosase o valor madeira. O mesmo ocorre coa terceira e carta columnas, almacenan Strings e amosan os seus valores no compoñente gráfico.

A continuación imos ver como é implementado o botón Informe dos mobles. Ao pulsar sobre este botón amósase toda a información almacenada sobre cada un dos obxectos seleccionados na táboa. seleccionados na táboa. Este é o código asociado ao botón:

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

        // Comprobar se hai mobles na taboa
        if(modeloMobles.getRowCount()==0)
        {
            JOptionPane.showMessageDialog(this, "Non hai mobles dispoñibles");
            return;
        }

        //Comprobar se seleccionamos algun moble
        if(tblMoblesDisponhibles.getSelectedRowCount()==0)
        {
            JOptionPane.showMessageDialog(this, "Debe seleccionar ao menos un moble");
            return;
        }

        //Recuperar os mobles do modelo

        int posicionesMobles[]=tblMoblesDisponhibles.getSelectedRows();

        String resultado="";
        for(int i=0;i<posicionesMobles.length;i++)
        {
            Moble moble=(Moble)modeloMobles.getValueAt(posicionesMobles[i], 0);
            resultado+="Nome: "+moble.getNome()+" Cor: "+moble.getCor()+
                    " Material: "+moble.getMaterial()+" Ancho: "+moble.getAncho()+
                    " Alto: "+moble.getAlto()+" Fondo: "+moble.getFondo()+"\n";

        }

        JOptionPane.showMessageDialog(this, resultado);
    }
```

O primeiro que temos que facer é comprobar se a táboa está baleira, xa que de ser así, non teremos información que amosar. Para elo executamos a seguinte liña:

```
        if(modeloMobles.getRowCount()==0)
```

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

A continuación comprobamos se o usuario seleccionou algunha fila da táboa. Para elo executamos a seguinte liña:

```
        if(tblMoblesDisponhibles.getSelectedRowCount()==0)
```

O método getSelectedRowCount aplícase sobre o compoñente táboa e devolve o número de filas seleccionadas na táboa. No caso de que devolva -1 quere dicir que non hai ningunha fila seleccionada.

A continuación temos que recuperar cales son as filas que o usuario seleccionou na táboa. Para elo executamos a seguinte liña:

```
        int posicionesMobles[]=tblMoblesDisponhibles.getSelectedRows();
```

O método getSelectedRows aplícase sobre o compoñente táboa e devolve un arrai de int que conten os índices das filas marcadas na táboa (a primeira fila é a cero). Como a táboa é de selección múltiple hai que empregar este método para recuperar tódolos elementos marcados. Se fora de selección simple, poderíamos empregar este método o tamén o método getSelectedRow.

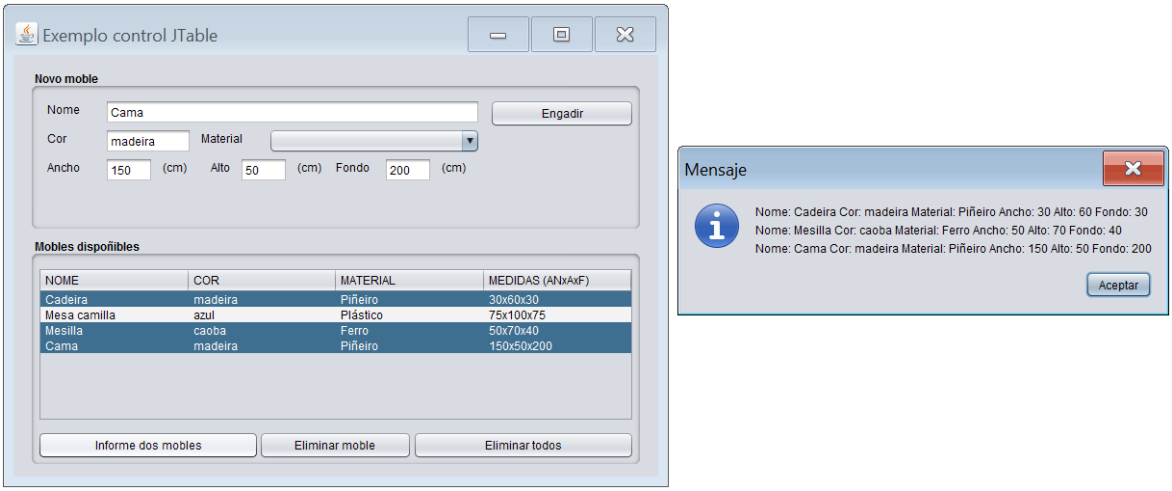
Finalmente imos recuperar tódolos elementos seleccionados na táboa. Para elo empregamos o método getValueAt, o cal aplícase sobre o almacén de datos que conten a información:

```
            Moble moble=(Moble)modeloMobles.getValueAt(posicionesMobles[i], 0);
```

Como pódese observar, para recuperar un valor do almacén de datos hai que indicar a súa posición dentro del. Isto faise pasando a súa fila e a súa columna. Neste caso a fila conseguímola do arrai de filas marcadas, mentres que a columna é a cero, xa que na columna

cero estamos almacenando o obxecto completo. Agora que foi recuperado un obxecto da clase Moble podemos acceder aos seus atributos a través de os seus métodos getter e setter.

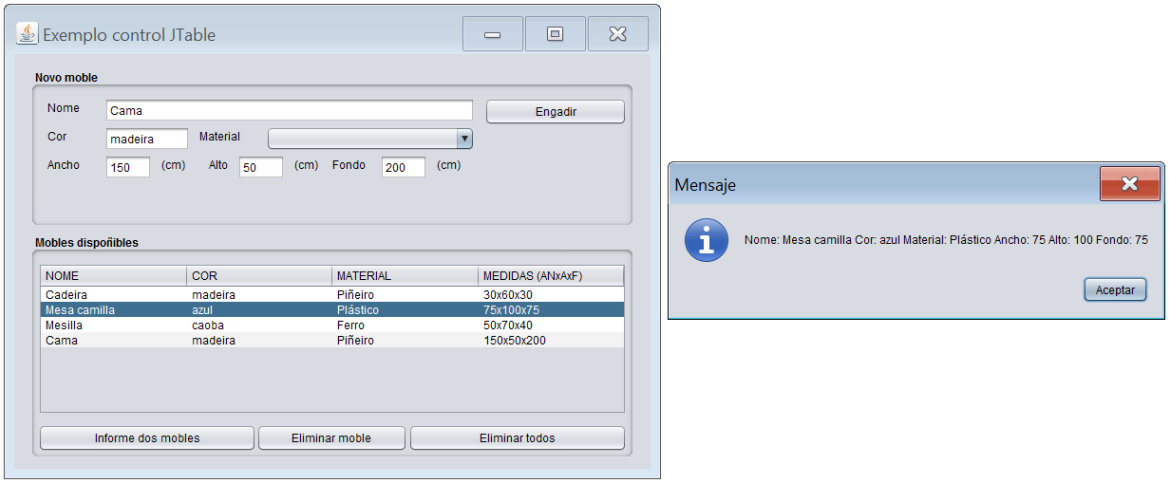
A continuación amósase o resultado de premer sobre o botón Informe dos mobles:



Outro dos requisitos da nosa aplicación é que ao facer dobre clic sobre algún elemento da táboa sexa amosada a información ao respecto dese elemento. Para elo, temos que implementar o evento mouseClicked da táboa:

```
    private void tblMoblesDisponhiblesMouseClicked(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:
        if(evt.getClickCount()==2)
        {
            //Recuperar os moble seleccionado
            Moble moble=(Moble)modeloMobles.getValueAt(tblMoblesDisponhibles.getSelectedRow(), 0);
            String resultado="Nome: "+moble.getNome()+" Cor: "+moble.getCor()+
                        " Material: "+moble.getMaterial()+" Ancho: "+moble.getAncho()+
                        " Alto: "+moble.getAlto()+" Fondo: "+moble.getFondo();
            JOptionPane.showMessageDialog(this, resultado);
        }
    }
```

A detección do dobre clic xa foi analizada cando vimos as listas. Respecto ao xeito de recuperar a información da fila sobre a que fixemos dobre clic, é similar ao empregado no botón Informe dos mobles visto anteriormente. A continuación amósase o resultado de facer dobre clic sobre un elemento da táboa:



O seguinte botón que imos ver como implementar é o botón Eliminar moble. Ao pulsar sobre este botón elimínanse os mobles seleccionados na táboa. Unha vez realizadas as validacións, a liña que se encarga de eliminar cada un dos elementos seleccionados na táboa é a seguinte:

```
modeloMobles.removeRow(posicionesMobles[i]);
```

O método removeRow aplícase sobre o almacén de datos. Mediante este método elimínanse tódolos obxectos almacenados na fila que se pasa como parámetro no almacén de datos.
Debido á ligazón establecida entre o almacén de datos e o compoñente gráfico táboa, ao eliminar o obxecto do almacén de datos, tamén desaparece da táboa.
O último botón que imos ver como implementar é o botón Eliminar todos. Ao pulsar sobre este botón elimínanse tódolos mobles da táboa. Unha vez realizadas as validacións, a liña que se encarga de eliminar tódolos elementos da táboa é a seguinte:

```
modeloMobles.setRowCount(0);
```

O xeito de eliminar tódolos elementos do almacén de datos é empregar o seu método setRowCount para indicar que queremos que a partir de agora o modelo de datos teña cero filas, é dicir, que quede baleiro. Debido á ligazón establecida entre o almacén de datos e o compoñente gráfico táboa, ao eliminar tódolos obxectos do almacén de datos, tamén desaparecen da táboa.

A continuación amósase o listado do código fonte desenvolvido para a realización da aplicación:

FrmPrincipal

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

import java.awt.event.KeyEvent;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;

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

    /**
     * Creates new form FrmPrincipal
     */
    public FrmPrincipal() {
        initComponents();
        scpMoblesDisponhibles.setVisible(false);
        //recuperar o modelo da taboa xerada a partir do entorno
        modeloMobles=(DefaultTableModel)tblMoblesDisponhibles.getModel();
        txtAncho.setDocument(new LimiteLonxitudeJTextField(5));
        txtAlto.setDocument(new LimiteLonxitudeJTextField(5));
        txtFondo.setDocument(new LimiteLonxitudeJTextField(5));
    }

    /**
     * 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() {

        pnlNovoMoble = new javax.swing.JPanel();
        lblNome = new javax.swing.JLabel();
        lblMaterial = new javax.swing.JLabel();
        lblAncho = new javax.swing.JLabel();
        lblAlto = new javax.swing.JLabel();
        txtAlto = new javax.swing.JTextField();
        txtNome = new javax.swing.JTextField();
        txtCor = new javax.swing.JTextField();
        txtAncho = new javax.swing.JTextField();
        lblCmAncho = new javax.swing.JLabel();
        lblCmAlto = new javax.swing.JLabel();
        btnEngadir = new javax.swing.JButton();
        lblFondo = new javax.swing.JLabel();
        txtFondo = new javax.swing.JTextField();
        lblCmFondo = new javax.swing.JLabel();
        lblCor = new javax.swing.JLabel();
        cmbMaterial = new javax.swing.JComboBox();
        pnlMoblesDisponhibles = new javax.swing.JPanel();
        btnEliminarTodos = new javax.swing.JButton();
        btnInformeMobles = new javax.swing.JButton();
```

```
        btnEliminarMoble = new javax.swing.JButton();
        lblNonDisponhibilidade = new javax.swing.JLabel();
        scpMoblesDisponhibles = new javax.swing.JScrollPane();
        tblMoblesDisponhibles = new javax.swing.JTable();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Exemplo control JTable");
        getContentPane().setLayout(null);

        pnlNovoMoble.setBorder(javax.swing.BorderFactory.createTitledBorder("Novo moble"));
        pnlNovoMoble.setLayout(null);

        lblNome.setText("Nome");
        pnlNovoMoble.add(lblNome);
        lblNome.setBounds(20, 30, 50, 20);

        lblMaterial.setText("Material");
        pnlNovoMoble.add(lblMaterial);
        lblMaterial.setBounds(180, 60, 60, 20);

        lblAncho.setText("Ancho");
        pnlNovoMoble.add(lblAncho);
        lblAncho.setBounds(20, 90, 45, 20);

        lblAlto.setText("Alto");
        pnlNovoMoble.add(lblAlto);
        lblAlto.setBounds(190, 90, 30, 20);

        txtAlto.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                txtAltoKeyTyped(evt);
            }
        });
        pnlNovoMoble.add(txtAlto);
        txtAlto.setBounds(220, 90, 50, 26);
        pnlNovoMoble.add(txtNome);
        txtNome.setBounds(80, 30, 390, 26);
        pnlNovoMoble.add(txtCor);
        txtCor.setBounds(80, 60, 90, 26);

        txtAncho.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                txtAnchoKeyTyped(evt);
            }
        });
        pnlNovoMoble.add(txtAncho);
        txtAncho.setBounds(80, 90, 50, 26);

        lblCmAncho.setText("(cm)");
        pnlNovoMoble.add(lblCmAncho);
        lblCmAncho.setBounds(140, 90, 40, 20);

        lblCmAlto.setText("(cm)");
        pnlNovoMoble.add(lblCmAlto);
        lblCmAlto.setBounds(280, 90, 40, 20);

        btnEngadir.setText("Engadir");
        btnEngadir.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btnEngadirActionPerformed(evt);
            }
        });
        pnlNovoMoble.add(btnEngadir);
        btnEngadir.setBounds(480, 30, 150, 29);

        lblFondo.setText("Fondo");
        pnlNovoMoble.add(lblFondo);
        lblFondo.setBounds(320, 90, 50, 20);

        txtFondo.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                txtFondoKeyTyped(evt);
            }
        });
        pnlNovoMoble.add(txtFondo);
        txtFondo.setBounds(370, 90, 50, 26);

        lblCmFondo.setText("(cm)");
        pnlNovoMoble.add(lblCmFondo);
        lblCmFondo.setBounds(430, 90, 40, 20);

        lblCor.setText("Cor");
        pnlNovoMoble.add(lblCor);
        lblCor.setBounds(20, 60, 25, 20);

            cmbMaterial.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Carballo", "Cerdeira", "Faia", "Ferro",
"Piñeiro", "Plástico" }));
        cmbMaterial.setSelectedIndex(-1);
        pnlNovoMoble.add(cmbMaterial);
        cmbMaterial.setBounds(250, 60, 220, 26);

        getContentPane().add(pnlNovoMoble);
        pnlNovoMoble.setBounds(15, 16, 640, 170);

        pnlMoblesDisponhibles.setBorder(javax.swing.BorderFactory.createTitledBorder("Mobles dispoñibles"));
        pnlMoblesDisponhibles.setLayout(null);

        btnEliminarTodos.setText("Eliminar todos");
        btnEliminarTodos.setEnabled(false);
        btnEliminarTodos.addActionListener(new java.awt.event.ActionListener() {
```

```
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btnEliminarTodosActionPerformed(evt);
            }
        });
        pnlMoblesDisponhibles.add(btnEliminarTodos);
        btnEliminarTodos.setBounds(400, 200, 230, 29);

        btnInformeMobles.setText("Informe dos mobles");
        btnInformeMobles.setEnabled(false);
        btnInformeMobles.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btnInformeMoblesActionPerformed(evt);
            }
        });
        pnlMoblesDisponhibles.add(btnInformeMobles);
        btnInformeMobles.setBounds(10, 200, 230, 29);

        btnEliminarMoble.setText("Eliminar moble");
        btnEliminarMoble.setEnabled(false);
        btnEliminarMoble.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btnEliminarMobleActionPerformed(evt);
            }
        });
        pnlMoblesDisponhibles.add(btnEliminarMoble);
        btnEliminarMoble.setBounds(240, 200, 159, 29);

        lblNonDisponhibilidade.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        lblNonDisponhibilidade.setText("NON HAI MOBLES DISPOÑIBLES");
        pnlMoblesDisponhibles.add(lblNonDisponhibilidade);
        lblNonDisponhibilidade.setBounds(0, 80, 640, 20);

        tblMoblesDisponhibles.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {

            },
            new String [] {
                "NOME", "COR", "MATERIAL", "MEDIDAS (ANxAxF)"
            }
        ) {
            boolean[] canEdit = new boolean [] {
                false, false, false, false
            };

            public boolean isCellEditable(int rowIndex, int columnIndex) {
                return canEdit [columnIndex];
            }
        });
        tblMoblesDisponhibles.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                tblMoblesDisponhiblesMouseClicked(evt);
            }
        });
        scpMoblesDisponhibles.setViewportView(tblMoblesDisponhibles);

        pnlMoblesDisponhibles.add(scpMoblesDisponhibles);
        scpMoblesDisponhibles.setBounds(10, 30, 620, 160);

        getContentPane().add(pnlMoblesDisponhibles);
        pnlMoblesDisponhibles.setBounds(15, 190, 640, 240);

        java.awt.Dimension screenSize = java.awt.Toolkit.getDefaultToolkit().getScreenSize();
        setBounds((screenSize.width-692)/2, (screenSize.height-501)/2, 692, 501);
    }// </editor-fold>

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

        //Validacion do formulario e recollida de datos
        String nome=txtNome.getText().trim();
        if(nome.compareTo("")==0)
        {
            JOptionPane.showMessageDialog(this, "Debe indicar o nome do moble");
            return;
        }

        String cor=txtCor.getText().trim();
        if(cor.compareTo("")==0)
        {
            JOptionPane.showMessageDialog(this, "Debe indicar a cor do moble");
            return;
        }

        if(cmbMaterial.getSelectedIndex()==-1)
        {
            JOptionPane.showMessageDialog(this, "Debe indicar o material do moble");
            return;
        }
        String material=(String)cmbMaterial.getModel().getElementAt(cmbMaterial.getSelectedIndex());

        String anchoS=txtAncho.getText().trim();
        if(anchoS.compareTo("")==0)
        {
            JOptionPane.showMessageDialog(this, "Debe indicar o ancho do moble en cm");
            return;
        }
        int ancho=new Integer(anchoS).intValue();

        String altoS=txtAlto.getText().trim();
        if(altoS.compareTo("")==0)
```

```
        {
            JOptionPane.showMessageDialog(this, "Debe indicar o alto do moble en cm");
            return;
        }
        int alto=new Integer(altoS).intValue();

        String fondoS=txtFondo.getText().trim();
        if(fondoS.compareTo("")==0)
        {
            JOptionPane.showMessageDialog(this, "Debe indicar o fondo do moble en cm");
            return;
        }
        int fondo=new Integer(fondoS).intValue();

        //Gardar datos nun obxeto da clase Moble
        Moble moble=new Moble(nome, cor, material, ancho, alto, fondo);
        //Visualizar taboa
        if(!scpMoblesDisponhibles.isVisible())
        {
            visualizarTaboaEBotons(true);
        }

        //Engadir o moble á táboa
        modeloMobles.setRowCount(modeloMobles.getRowCount()+1);
        modeloMobles.setValueAt(moble, modeloMobles.getRowCount()-1 , 0);
        modeloMobles.setValueAt(moble.getCor(), modeloMobles.getRowCount()-1 , 1);
        modeloMobles.setValueAt(moble.getMaterial(), modeloMobles.getRowCount()-1 , 2);
        String medidas=moble.getAncho()+"x"+moble.getAlto()+"x"+moble.getFondo();
        modeloMobles.setValueAt(medidas, modeloMobles.getRowCount()-1 , 3);

        //limpar o formulario
        txtNome.setText("");
        txtCor.setText("");
        txtAncho.setText("");
        txtAlto.setText("");
        txtFondo.setText("");
        cmbMaterial.setSelectedIndex(-1);

    }

    private void txtAnchoKeyTyped(java.awt.event.KeyEvent evt) {
        // TODO add your handling code here:
        if(!isNumero(evt))
        {
            evt.consume();
        }
    }

    private void txtAltoKeyTyped(java.awt.event.KeyEvent evt) {
        // TODO add your handling code here:
        if(!isNumero(evt))
        {
            evt.consume();
        }
    }

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

        // Comprobar se hai mobles na taboa
        if(modeloMobles.getRowCount()==0)
        {
            JOptionPane.showMessageDialog(this, "Non hai mobles dispoñibles");
            return;
        }

        //Comprobar se seleccionamos algun moble
        if(tblMoblesDisponhibles.getSelectedRowCount()==0)
        {
            JOptionPane.showMessageDialog(this, "Debe seleccionar ao menos un moble");
            return;
        }

        //Recuperar os mobles do modelo

        int posicionesMobles[]=tblMoblesDisponhibles.getSelectedRows();

        String resultado="";
        for(int i=0;i<posicionesMobles.length;i++)
        {
            Moble moble=(Moble)modeloMobles.getValueAt(posicionesMobles[i], 0);
            resultado+="Nome: "+moble.getNome()+" Cor: "+moble.getCor()+
                    " Material: "+moble.getMaterial()+" Ancho: "+moble.getAncho()+
                    " Alto: "+moble.getAlto()+" Fondo: "+moble.getFondo()+"\n";

        }

        JOptionPane.showMessageDialog(this, resultado);
    }

    private void btnEliminarMobleActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        // Comprobar se hai mobles na taboa
        if(modeloMobles.getRowCount()==0)
        {
            JOptionPane.showMessageDialog(this, "Non hai mobles dispoñibles");
            return;
        }

        if(tblMoblesDisponhibles.getSelectedRowCount()==0)
```

```
        {
            JOptionPane.showMessageDialog(this, "Debe seleccionar ao menos un moble");
            return;
        }

        //Eliminar os  mobles seleccionados do modelo
        int posicionesMobles[]=tblMoblesDisponhibles.getSelectedRows();
        for(int i=posicionesMobles.length-1;i>=0;i--)
        {
            modeloMobles.removeRow(posicionesMobles[i]);
        }

        if(modeloMobles.getRowCount()==0) visualizarTaboaEBotons(false);
    }

    private void visualizarTaboaEBotons(boolean visualizar)
    {
        if(visualizar)
        {
            lblNonDisponhibilidade.setVisible(false);
            scpMoblesDisponhibles.setVisible(true);
            btnInformeMobles.setEnabled(true);
            btnEliminarMoble.setEnabled(true);
            btnEliminarTodos.setEnabled(true);
        }
        else
        {
            scpMoblesDisponhibles.setVisible(false);
            btnInformeMobles.setEnabled(false);
            btnEliminarMoble.setEnabled(false);
            btnEliminarTodos.setEnabled(false);
            lblNonDisponhibilidade.setVisible(true);
        }
    }

    private void btnEliminarTodosActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        // Comprobar se hai mobles na taboa
        if(modeloMobles.getRowCount()==0)
        {
            JOptionPane.showMessageDialog(this, "Non hai mobles dispoñibles");
            return;
        }

        //Eliminar tódolos mobles da taboa
        modeloMobles.setRowCount(0);
        visualizarTaboaEBotons(false);
    }

    private void txtFondoKeyTyped(java.awt.event.KeyEvent evt) {
        // TODO add your handling code here:
        if(!isNumero(evt))
        {
            evt.consume();
        }
    }

    private void tblMoblesDisponhiblesMouseClicked(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:
        if(evt.getClickCount()==2)
        {
            //Recuperar os moble seleccionado
            Moble moble=(Moble)modeloMobles.getValueAt(tblMoblesDisponhibles.getSelectedRow(), 0);
            String resultado="Nome: "+moble.getNome()+" Cor: "+moble.getCor()+
                        " Material: "+moble.getMaterial()+" Ancho: "+moble.getAncho()+
                        " Alto: "+moble.getAlto()+" Fondo: "+moble.getFondo();
            JOptionPane.showMessageDialog(this, resultado);
        }
    }

    private boolean isNumero(java.awt.event.KeyEvent evt)
    {
        // Comprobar se a pulsacion de teclado pasada
        // en evt é un número
        char caracter=evt.getKeyChar();
        if((
            (caracter<'0' || caracter >'9')&&
            //(caracter!=KeyEvent.VK_SPACE) &&
            (caracter!=KeyEvent.VK_BACK_SPACE)
            ) ||
            (evt.isAltDown() ||
             evt.isAltGraphDown() ||
             evt.isControlDown() ||
             evt.isMetaDown()
           ))
        {
            return false;
        }
        return true;
    }

    /**
     * @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 btnEliminarMoble;
    private javax.swing.JButton btnEliminarTodos;
    private javax.swing.JButton btnEngadir;
    private javax.swing.JButton btnInformeMobles;
    private javax.swing.JComboBox cmbMaterial;
    private javax.swing.JLabel lblAlto;
    private javax.swing.JLabel lblAncho;
    private javax.swing.JLabel lblCmAlto;
    private javax.swing.JLabel lblCmAncho;
    private javax.swing.JLabel lblCmFondo;
    private javax.swing.JLabel lblCor;
    private javax.swing.JLabel lblFondo;
    private javax.swing.JLabel lblMaterial;
    private javax.swing.JLabel lblNome;
    private javax.swing.JLabel lblNonDisponhibilidade;
    private javax.swing.JPanel pnlMoblesDisponhibles;
    private javax.swing.JPanel pnlNovoMoble;
    private javax.swing.JScrollPane scpMoblesDisponhibles;
    private javax.swing.JTable tblMoblesDisponhibles;
    private javax.swing.JTextField txtAlto;
    private javax.swing.JTextField txtAncho;
    private javax.swing.JTextField txtCor;
    private javax.swing.JTextField txtFondo;
    private javax.swing.JTextField txtNome;
    // End of variables declaration
    private DefaultTableModel modeloMobles;
}
```

Moble

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

/**
 *
 * @author German DR
 */
public class Moble {
    private String nome;
    private String cor;
    private String material;
    private int ancho;
    private int alto;
    private int fondo;

    public Moble(String nome, String cor, String material, int ancho, int alto, int fondo) {
        this.nome = nome;
        this.cor = cor;
        this.material = material;
        this.ancho = ancho;
        this.alto = alto;
        this.fondo = fondo;
    }

    @Override
    public String toString() {
        return nome;
    }

    public String getCor() {
        return cor;
    }

    public String getMaterial() {
        return material;
    }

    public int getAncho() {
        return ancho;
```

```
    }

    public int getAlto() {
        return alto;
    }

    public int getFondo() {
        return fondo;
    }

    public String getNome() {
        return nome;
    }

}
```

LimiteLonxitudeJTextField

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

/**
 *
 * @author German DR
 */
import javax.swing.text.AttributeSet;
import javax.swing.text.BadLocationException;
import javax.swing.text.PlainDocument;

public class LimiteLonxitudeJTextField extends PlainDocument
{
  private int lonxitude;

  public LimiteLonxitudeJTextField(int lonxitude)
  {
   super();
   this.lonxitude = lonxitude;
  }

  @Override
  public void insertString( int offset, String  str, AttributeSet attr ) throws BadLocationException
  {
    if (str == null) return;

    if ((getLength() + str.length()) <= lonxitude)
    {
      super.insertString(offset, str, attr);
    }
  }
}
```