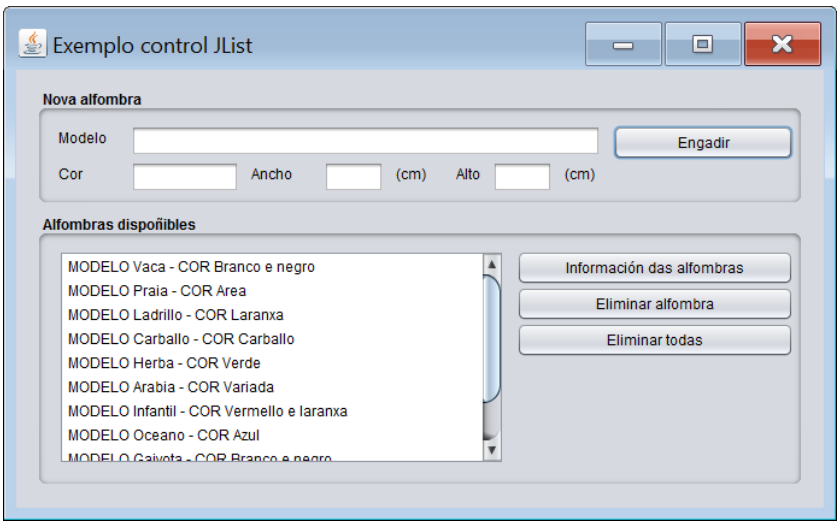
1. Xestión de compoñentes avanzados

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 nos serven 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

1.1.1.1 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 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 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

Eventos do control JList empregados máis habitualmente e que se poden 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.

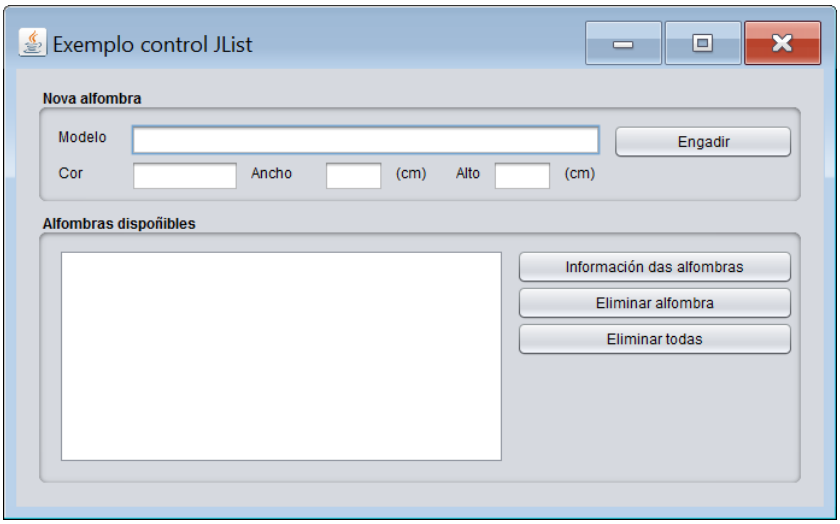
ValueChanged	Este evento é disparado cando cambiamos o elemento seleccionado na lista.

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 multi-selección.

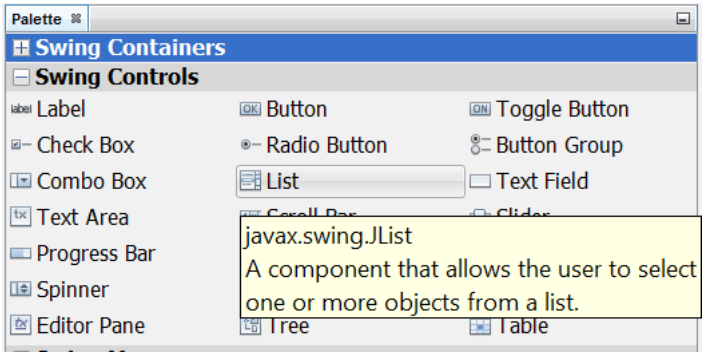
Xestión de listas empregando NetBeans.

A continuación imos desenvolver o seguinte formulario:

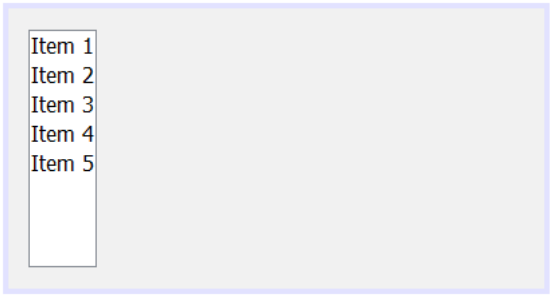


O formulario consta dunha lista, a cal se atopa no panel de alfombras dispoñibles. Será empregado para engadir nela as alfombras que definamos desde o panel nova alfombra. A través dos tres botóns adxuntos á lista poderemos realizar diferentes accións sobre os elementos da lista. Ademais, no caso de 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:



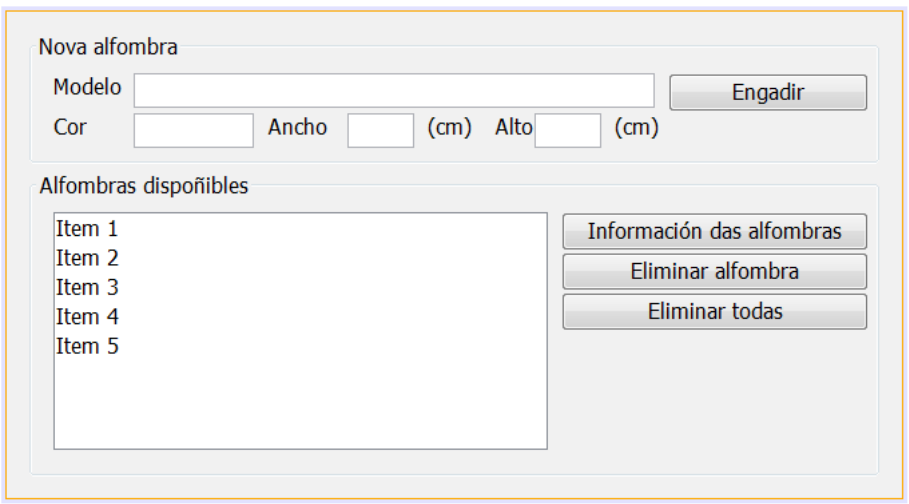
e arrastrámolo ata o formulario:



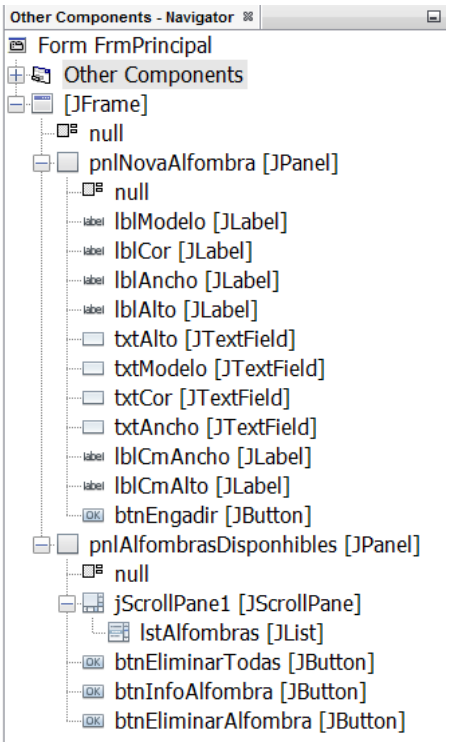
Cando arrastramos unha lista ao noso formulario en realidade créanse dous compoñentes: un JScrollPane e dentro deste a lista. Isto é debido a que cando engadimos elementos dentro do compoñente lista, pode ocorrer que haia máis elementos na lista dos que se poden visualizar nela. Para desprazarnos polos 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:

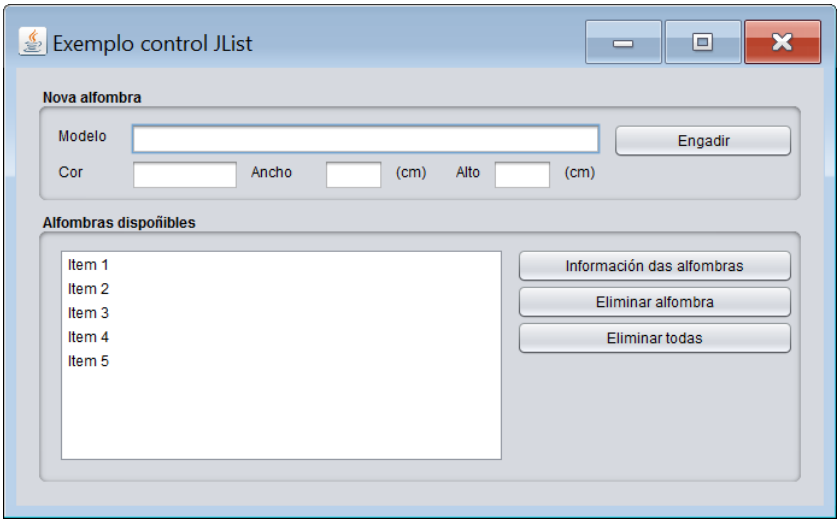


O noso formulario quedará configurado do seguinte xeito:



Unha vez que temos colocados os compoñentes que necesitamos hai que configurar o seu aspecto, pero afortunadamente 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 se pode observar o comportamento visual da aplicación xa esta resolto (unicamente temos o pequeno detalle dos elementos que se amosan na lista, pero ímolo resolver en breve). Agora imos desenvolver a súa funcionalidade.

Anteriormente, mencionamos que o funcionamento da lista 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 amosado non ten que coincidir coa información almacenada internamente, senón que é o programador o que pode decidir que é o que se amosa no

compoñente gráfico. Segundo o que acabamos de explicar, para empregar o paradigma vista controlador necesitamos dous elementos, o compoñente gráfico (lstAlfombras) 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.DefaultListModel. O seu funcionamento é practicamente idéntico ao dun java.util.Vector. 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.DefaultListModel<Alfombra> modeloAlfombras;
```

Como se pode observar o obxecto modeloAlfombra (<Alfombra>) foi tipado xa que os obxectos que imos almacenar van ser todos de clase Alfombra (a clase Alfombra define as características que ten unha alfombra. Ver código) e deste xeito evitámonos o casteado dos obxectos, pero non é algo obrigatorio (de feito, cando describamos o obxecto JComboBox, desenvolveremos un programa sin empregar casteado co fin de ter os dous puntos de vista).

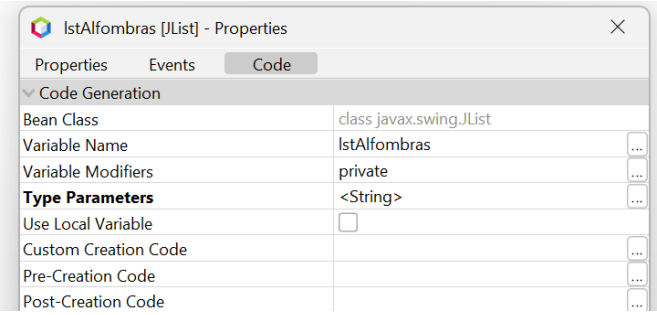
Unha vez definida a estrutura de datos que vai funcionar como almacén dos datos da lista, temos que ligala ao compoñente gráfico de tipo lista que vai amosar os seus datos. Isto facémolo nalgún punto do código que se execute antes de que o usuario poida empezar a interaccionar co formulario. Quizais,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

```
public FrmPrincipal() {
initComponents();
//crear o modelo para almacenar as alfombras
modeloAlfombras=new DefaultListModel<Alfombra>();
//ligar o modelo ao control JList
lstAlfombras.setModel(modeloAlfombras);
txtAncho.setDocument(new LimiteLonxitudeJTextField(5));
txtAlto.setDocument(new LimiteLonxitudeJTextField(5));
}
```

A seguinte liña:

```
modeloAlfombras=new DefaultListModel<Alfombra>();
```

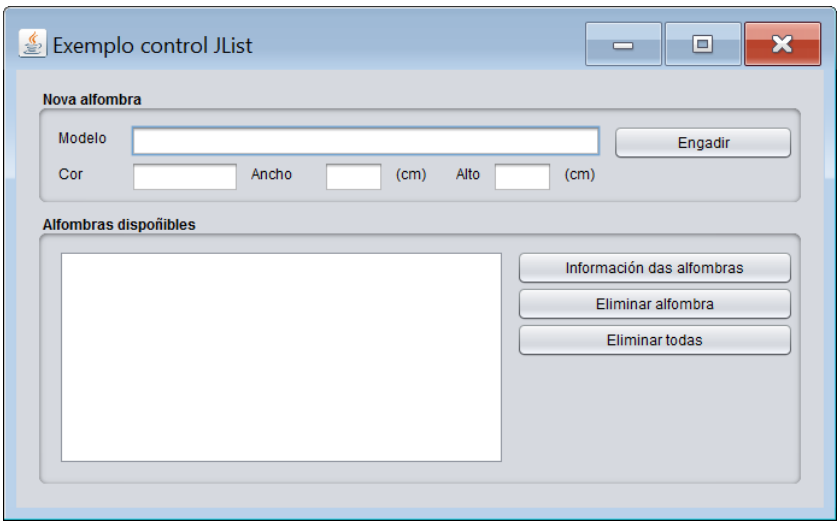
empregámola para reservar memoria para o almacén de datos (opcionalmente poderíase facer na propia declaración).



Deberemos eliminar o tipo de parámetro <String> que ven por defecto xa que coa liña:

```
lstAlfombras.setModel(modeloAlfombras);
```

ligamos o almacén de datos ao compoñente gráfico e establecemos o modelo axeitado. Deberiamos eliminar á súa vez os valores cargados en model das propiedades do compoñente. Unha vez ligado o almacén de datos ao compoñente gráfico, cando executemos a aplicación no compoñente gráfico amosarase o contido do almacén de datos, o cal inicialmente está baleiro:



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

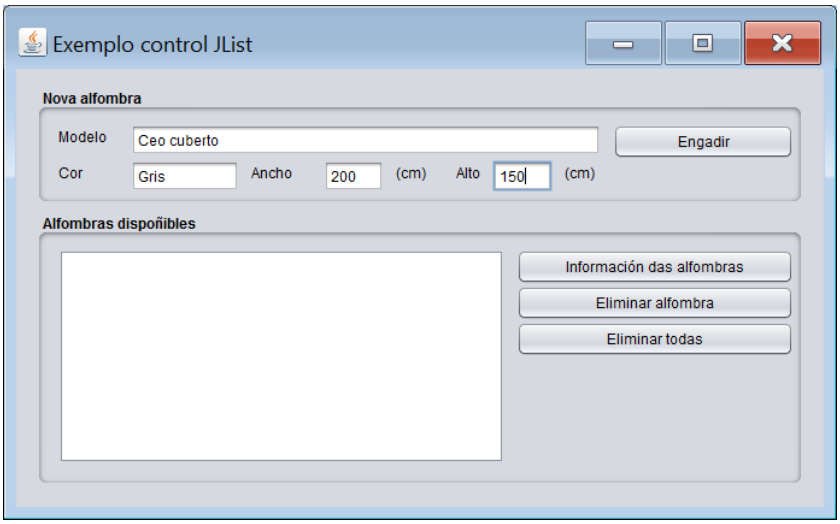
```
modeloAlfombras.addElement(alfombra);
```

Con esta liña estamos introducindo o obxecto alfombra (de clase Alfombra) na estrutura de datos ligada ao compoñente gráfico. Automaticamente amosarase o obxecto no compoñente lista, pero, ¿que se amosará?. Ben, vaise amosar o que devolva o método toString do obxecto que acabamos de almacenar. Polo tanto, é responsabilidade do programador reescribir este método co fin de que devolva aquela información que desexa que se amose. No noso caso, o método toString da clase Alfombra é o seguinte:

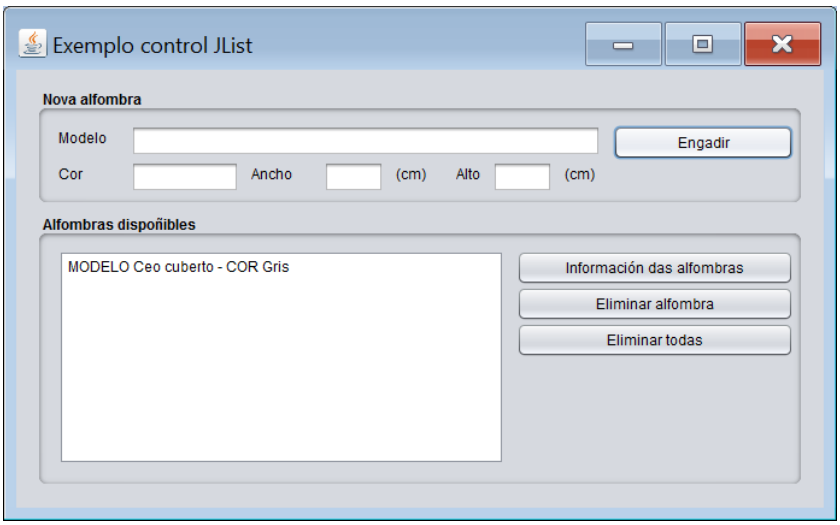
```
@Override
public String toString() {
return "MODELO " + modelo + " - COR " + cor;
    }
```

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

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 un obxecto de clase Alfombra cuxos atributos son modelo: Ceo cuberto, cor: Gris, ancho: 200 e alto 150, pero o compoñente gráfico unicamente amosa o valor que devolve a aplicación do método toString do obxecto de clase Alfombra.

A continuación imos implementar o botón Información das alfombras. Ao pulsar sobre este botón amosarase toda a información almacenada sobre cada un dos obxectos seleccionados na lista. 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 na lista
if(modeloAlfombras.getSize()==0)
        {
JOptionPane.showMessageDialog(this, "Non hai alfombras dispoñibles");
return;
        }

//Comprobar se seleccionamos algunha alfombra
if(lstAlfombras.getSelectedIndex()==-1)
        {
JOptionPane.showMessageDialog(this, "Debe seleccionar ao menos unha alfombra");
return;
        }

//Recuperar as alfombras do meodelo
int posicionsSeleccionados[]=lstAlfombras.getSelectedIndices();
String mensaxe="";
for(int i=0;i<posicionsSeleccionados.length;i++)
        {
Alfombra alfombra=modeloAlfombras.getElementAt(posicionsSeleccionados[i]);
mensaxe=mensaxe+"MODELO: "+alfombra.getModelo()+"\nCOR: "+alfombra.getCor()+
"\nANCHO: "+alfombra.getAncho()+" cm\nALTO: "+alfombra.getAlto()+" cm\n\n";
        }


//Mostras información das alfombra por pantalla
JOptionPane.showMessageDialog(this, mensaxe);

    }
```

O primeiro que temos que facer é comprobar se a lista está baleira, xa que de ser así, non teremos 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 contén a información. Devólvenos o número de elementos que contén o almacén de datos.

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

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

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

A continuación temos que recuperar cales son os elementos que o usuario seleccionou na lista. Para elo executamos a seguinte liña:

```
int posicionsSeleccionados[]=lstAlfombras.getSelectedIndices();
```

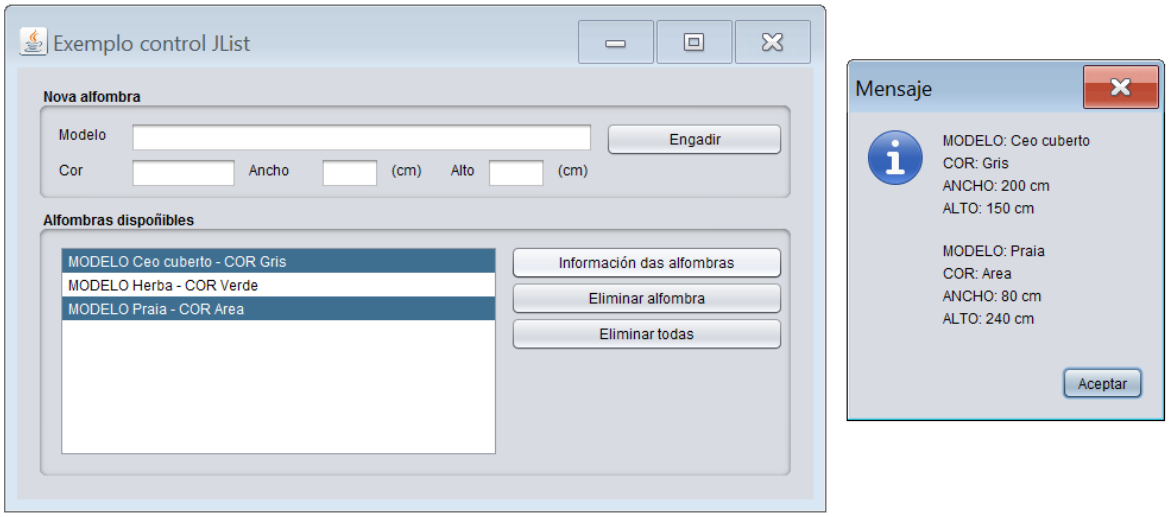
O método getSelectedIndices aplícase sobre o compoñente lista e devolve un array de int que contén os índices dos elementos marcados na lista. Como a lista é de selección múltiple hai que empregar este método para recuperar tódolos elementos marcados. Se fora de selección simple, poderíamos empregar este método o tamén o getSelectedIndex.

Finalmente imos recuperar tódolos elementos seleccionados na lista. Para elo empregamos o método getElementAt, que se aplica sobre a estrutura de datos que contén a información:

```
Alfombra alfombra=modeloAlfombras.getElementAt(posicionsSeleccionados[i]);
```

Como se pode observar unicamente hai que pasarlle como parámetro a posición do elemento que queremos recuperar. Como no seu momento declaramos a modeloAlfombras con un tipado <Alfombra>, non será necesario realizar ningún casteado e polo tanto o valor devolto polo método getElementAt asígnase directamente a unha variable de clase Alfombra. Agora 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 das alfombras:



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

```
private void lstAlfombrasMouseClicked(java.awt.event.MouseEvent evt) {
// TODO add your handling code here:
if(evt.getClickCount()==2)
        {
// Comprobar se hai alfombras na lista
if(modeloAlfombras.getSize()==0)
            {
JOptionPane.showMessageDialog(this, "Non hai alfombras dispoñibles");
return;
            }

//Comprobar se seleccionamos algunha alfombra. Dependendo da version da maquina
//virtual esta comprobación pódese obviar xa que nalgunhas versións, ao facer clic
//sobre unha lista con elementos, se o clic non se fai sobre ningún deles
//automáticamente o entorno selecciona un por nos.
if(lstAlfombras.getSelectedIndex()==-1)
            {
JOptionPane.showMessageDialog(this, "Debe seleccionar unha alfombra");
return;
            }

//Recuperar a alfombra do modelo
Alfombra alfombra=modeloAlfombras.getElementAt(lstAlfombras.getSelectedIndex());
//Mostras información da alfombra por pantalla
String mensaxe="MODELO: "+alfombra.getModelo()+"\nCOR: "+alfombra.getCor()+
"\nANCHO: "+alfombra.getAncho()+" cm\nALTO: "+alfombra.getAlto()+" cm";
JOptionPane.showMessageDialog(this, mensaxe);
```

```
        }
    }
```

No código anterior cabe destacar como é realizada a detección do dobre clic:

```
if(evt.getClickCount()==2)
```

O evento mouseClicked execútase cando facemos clic sobre a lista. Ademais a través do parámetro recibido pódese conseguir información máis detallada acerca destas pulsacións do rato. En concreto o método getClickCount devólvenos o número de clics que fixeron saltar o evento (un clic, un dobre clic, un triple clic, ...). Neste caso estamos detectando un dobre clic. O resto do código non merece mención algunha xa que é moi similar ao desenvolvido para o botón Información das alfombras. A continuación amósase o resultado de facer dobre clic sobre un elemento da lista:

O seguinte botón que imos ver como implementar é o botón Eliminar alfombra. Ao pulsar sobre este botón elimínase a alfombra seleccionada na lista. Unha vez realizadas as validacións, a liña que se encarga de eliminar o elemento seleccionado na lista é a seguinte:

```
modeloAlfombras.removeElementAt(lstAlfombras.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 lista, ao eliminar o obxecto do almacén de datos, tamén desaparece da lista.

O último botón que imos implementar é o botón Eliminar todas. Ao pulsar sobre este botón elimínanse tódalas alfombra da lista. Unha vez realizadas as validacións, a liña que se encarga de eliminar tódolos elementos da lista 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 lista, ao eliminar tódolos obxectos do almacén de datos, tamén desaparecen da lista.