1 Swing

1.1 Sumario

1.2 Swing básico

Java ten 2 paquetes, *AWT (Abstract Windows Toolkit)* e *Swing*. *AWT* utiliza as rutinas de ventás orixinais do SO e por tanto os efectos visuais dependen do SO. Sen embargo *Swing* permite tres modos: *"look and feel"* de Java, a apariencia nativa do SO, ou outra apariencia. *Swing* está construido por enriba de *AWT*, deste xeito os compoñentes *Swing* van precedidos de **J** para distinguilos dos seus homónimos *AWT* (Ex.: *JFrame* en lugar de *Frame*). Para incluir os compoñentes *Swing* dentro de un proxecto deberemos importar os paquetes *java.awt.**, *java.awt.event.** e *javax.swing.**

1.3 Contenedores, marcos e paneis

Os contenedores son controis onde situaremos outros controis. *Frames* son contenedores de alto nivel tales como *JFrame, JWindow, JDialog, JApplet* ou *JInternalFrame* que interactuarán co xestor de ventás do SO. Os paneis son contenedores intermedios tales como *JPanel, JOptionPane, JScrollPane, JLayeredPane, JSplitPane* e *JTabbedPane* utilízanse para organizar a composición do resto dos controis dentro da ventá.

1.4 JFrame e JPanel

JFrame é o contenedor de alto nivel máis empregado. Engade funcionalidades básicas tales como minimizar, maximizar, pechar, o título da ventá e os bordes básicos. Algúns métodos importantes de *JFrame* son: *setBounds(x,y,w,h), setLocation(x,y), setSize(w,h), setResizable(bool), setTitle(str), setVisible(bool), isResizable() e getTitle()*. O método setDefaultCloseOperation(constante) controla a acción ao pulsar o x da ventá. Normalmente usarase a constante *JFrame.EXIT_ON_CLOSE*.

JPanel é o panel de contido máis común. Crearemos unha instancia do panel e engadirémola ao *JFrame*. O método *add()* permitirá engadir controis ao panel. A maneira en que estes controis se engaden ao panel ven definida polo *"layout manager"*.

O seguinte exemplo crea un *JFrame* e engádelle un *JPanel*.

```
import javax.swing.*;

public class Frame1 extends JFrame {
public Frame1() {
super("JFrame básico");
setBounds(100,100,300,100);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// Crea e un panel e asociao co frame
JPanel pane = new JPanel();
this.getContentPane().add(panel);

// Fai visible o frame
setVisible(true);
}
public static void main(String args[]) {
new Frame1();
}
```

```
}
```

1.5 Caixas de diálogo simples

Os dialogos son ventás con información sinxela, ou peticións de información simple que se mostran como *popups*. En *Swing* utilizaremos a clase *JOptionPane* onde estarán os métodos básicos de cada tipo de diálogo. A clase *JDialog* pode usarse para crear caixas de diálogo personalizadas xa que devolve unha ventá simple sen ningún tipo de adorno.

Cada método *JOptionPane* ten un primeiro parámetro apuntando ao pai (a ventá sobre a que aparecerá o diálogo) ou null (a ventá actual). O segundo parámetro será a mensaxe ou o *prompt* para mostrar.

O método *showMessageDialog()* ten dous parámetros opcionais co título do diálogo e o icono para mostrar. Este diálogo mostrará un só boton *Ok* e non se devolverá ningún tipo de dato.

```
JOptionPane.showMessageDialog(null,"Mensaxe simple para o usuario",
    "Diálogo da mensaxe",JOptionPane.PLAIN_MESSAGE);
```

O método *showConfirmDialog()* ten tres parámetros opcionais para establecer o título do diálogo, os botóns para mostrar e o icono. Por defecto haberá tres botóns: Si, Non e Cancelar. Os valores devoltos por este método son *JOptionPanel.YES_OPTION*, *JOptionPanel.NO_OPTION* e *JOptionPanel.CANCEL_OPTION*.

```
int pressed = JoptionPane.showConfirmDialog(null,"Todo ben?");
if (pressed==JOptionPane.YES_OPTION)
   {
        // Acción ao pulsar si
   }
```

O método *showInputDialog()* ten dous parámetros opcionais para establecer o título da ventá de diálogo e o icono. Haberá dous botóns: *Ok* e *Cancelar* para finalizar o diálogo. Calquera información que tecleemos dentro da ventá de diálogo será devolta como *String*.

```
String nomeUsuario = JOptionPane.showInputDialog(null,"Cal é o teu nome?");
```

A lista de tipos de iconos que se poden mostrar (usando constantes predefinidas) son: *ERROR_MESSAGE, INFORMATION_MESSAGE, PLAIN_MESSAGE, QUESTION_MESSAGE*, e *WARNING_MESSAGE*.

1.6 Etiquetas, iconos e botóns

As etiquetas mostrarán texto non interactivo e normalmente usaranse para mostrar información dentro da ventá. Para crear unha etiqueta usaremos a clase *JLabel* pasándolle o texto a mostrar como parámetro ao seu construtor. Outro parámetro que podemos pasarlle é o tipo de aliñamento horizontal. O aliñamento vertical establecese a partires do método *setVerticalAlignment()*. Podemos cambiar o contido da etiqueta usando o método *setText()*.

Podemos engadir iconos as etiquetas ou a calquera outro control para facelos máis vistosos. Para crear un icono usaremos a clase *IconImage*. Podemos usar un parámetro opcional para para controlar a posición do texto relativa ao icono.

```
ImageIcon icon = new ImageIcon("sorriso.gif");
JLabel label = new JLabel("Ola!",icon,SwingConstants.RIGHT);
panel.add(label);
```

Os botóns utilizanse para realizar operacións a petición do usuario. Para crear botóns usaremos a clase *Jbutton*. Podemos desactivalos co método *setEnabled(false)* e comprobar se están activados co método *isEnabled()*. Un método bastante usado é *setMnemonic(char)* que permite asociar unha tecla de acceso rápido co botón. Os botón precisan dun *ActionEvent*(é un *event listener*) que indicará a acción a realizar cando se prema nel.

```
import javax.swing.*;
public class Frame2 extends JFrame
{
public Frame2() {
super("Demo de botón");
setBounds(100,100,300,200);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

JPanel panel = new JPanel();
this.getContentPane().add(panel); // Panel con FlowLayout

JButton boton = new JButton("Preme aquí");
```

```
boton.setMnemonic('B'); // Tecla de acceso rápido ao botón

panel.add(boton);

boton.requestFocus();

setVisible(true);
}
public static void main(String args[]) {
new Frame2();
}
}
```

Os botóns con dous estados (*Toggle buttons*) mostran botóns que son como interruptores poden estar pulsados ou non. Crearanse coa clase JToggleButton. O método *isSelected()* devolverá o estado do botón. Ademáis do *ActionEvent* tamén se disparará o *ChangeEvent*.

1.7 Caixas de texto

As caixas de texto creanse usando a clase *JTextField*. *JPassWordField* representa unha caixa de texto para introducir contrasinais (mostrará asteriscos en lugar dos caracteres tecleados). Este tipo de controis utilízanse normalmente para mostrar información.

1.8 Áreas de texto

As áreas de texto son zonas de edición para bloques ou múltiples liñas de texto. Constrúense usando a clases JTextArea para texto simple, JEditorPane para para html, rtf e texto formatado en xeral. Os método máis útiles son getText(), setText(), append(str), insert(str,pos), replace(str, comezo, fin), setEditable(bool), setLineWrap(bool) e setWrapStyleWord(bool).

Con frecuencia o texto dentro dunha caixa de texto excede o tamaño da propia área de texto e por tanto faise necesario incluir scrolling. Para isto utilizaremos a clase JScrollPane pasándolle ao construtor da mesma o obxecto do que desexamos utilizar scroll. JTextPane edit = new JTextPane(); JScrollPane scroll = new JScrollPane(edit); panel.add(scroll); //JPanel panel = new JPanel();

1.9 Listas

As listas permiten que o usuario faga unha selección entre un lista de opcións predefinidas. Poden ser deplegables ou non. Para crear unha lista usaremos a clase *JList*. O seu construtor recibirá como parámetro o modelo da lista. As listas constrúense normalmente usando scroll (clase *JScrollPane*) para permitir que o usuario vexa todos os elementos da lista.

Os modelos de listas definen os métodos de xestión dos elementos da lista. A clase *DefaultListModel* contén o construtor *DefaultListModel(array)* e os seguintes métodos de xestión: *addElemento(nomObx), elementAt(index), removeElement(nomObx), removeAllElements(), removeElementAt(index), contains(momObx), indexOf(nome), size()* e un método de conversión dos elementos da lista nun array *copyInto(array)*. *getSelectedValue()* devolve o elemento seleccionado ou *getSelectedIndex()* devolve o número de índice do elemento seleccionado. Para permitir selección múltiple utilízase o método *setSelectionMode(2)*. A selección por defecto pode establecerse co método *setSelectedIndex(x)*. Cando permitimos selección múltiple podemos obter un array cos elementos seleccionados usando o método *getSelectedValues()*.

Os listas desplegables son un tipo especial de lista que crearemos usando a clase *JComboBox*. O método *setEditable(true)* fai os elementos do combo editables. O elemento seleccionado obtense cos métodos *getSelectedItem()* e *getSelectedIndex()*. Outros métodos útiles serán *getItemAt(idx), getItemCount(), setSelectedItem(obx)* e *setMaximunRowCount(x)*.

1.10 Táboas

As táboas representan unha das formas máis comúns de amosar un conxunto de datos relacionais, coma por exemplo os rexistros dunha base de datos.
Java incorpora unha clase de nome **JTable** no paquete **javax.swing** para permitir o acceso a un compoñente que poida manipular calquera tipo de táboa.

Para engadir unha táboa a un contenedor seguimos os pasos acostumados para engadir calquera compoñente a un contenedor:

1. Construimos o obxecto **JTable**
2. Establecemos as súas propiedades
3. Engadímolo ao contenedor, normalmente un formulario(marco JFrame) ou un panel (JPanel).

As táboas están compostas por filas e columnas e representan os datos en celdas, sendo unha celda a rexión formada pola intersección dunha fila e unha columna. Nas táboas poden existir 2 tipos de filas e columnas: editables e non editables. Ás veces o número de filas e columnas é superior á superficie da táboa co cal é necesario empregar barras de desprazamento nun panel de desprazamento (JScrollPane) para poder visualizar o contido de todas as filas e todas as columnas. É interesante saber que a barra de desprazamento horizontal soamente aparece se a propiedade *autoResizeMode* toma valor *OFF*:

```
obxecto_jtable.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);
```

1.10.1 O constructor dunha táboa

Existen varios constructores da clase JTable, un moi empregado é o que posúe 2 argumentos: un de tipo Object[][] que permite almacenar nunha matriz bidimensional as filas e columnas que amosará a táboa e outro de tipo String[] que permite almacenar nunha matriz unidimensional as cabeceiras das columnas.

```
//Declaración de variables

//Campo taboa1 tipo JTable
private javax.swing.JTable taboa1;

//Datos para a táboa nunha matriz bidimensional
Object[][] datos = {
        {"Xan", "22", new Long(11111111), new Boolean(true)} ,
        {"Pia", "32", new Long(12111111), new Boolean(true)} ,
        {"Xurxo", "27", new Long(13111111), new Boolean(false)}
} ;

//Cabeceiras da táboa nunha matriz unidimensional
String[] cabeceiras = { "Nome", "Idade", "Teléfono", "Casado" } ;

//Constructor da táboa
taboa1 = new javax.swing.JTable (datos, cabeceiras) ;
```

1.10.2 Barras de desprazamento nunha táboa

Para engadir barras de desprazamento a unha táboa empregamos un panel de desprazamento **JScrollPane**. Exemplo:

```
//Declaración de variables

//Creación do panel de desprazamento jscrollpane1 do tipo JScrollPane
private javax.swing.JScrollPane jscrollpane1 ;

//Campo taboa1 tipo JTable
private javax.swing.JTable taboa1;

//Engadir na táboa taboa1 as barras de desprazamento do panel de desprazamento jscrollpane1
jscrollpane1 = new javax.swing.JScrollPane(taboa1) ;
```

1.10.3 Modelos dunha táboa

Unha táboa posúe os seguintes modelos:

- **TableModel**: Modelo que dá información de filas, columnas e celdas. **JFC** proporciona unha implementación estándar da interface **TableModel** baixo a clase **DefaultTableModel** derivada de **AbstractTableModel**
- **TableColumnModel**: Modelo para almacenar varias columnas como unha colección. **JFC** proporciona unha implementación estándar da interface **TableColumnModel** baixo a clase **DefaultTableColumnModel**

Unha táboa implementa por omisión un modelo, isto é, a súa propiedade *model* fai referencia a un obxecto da clase **DefaultTableModel**, implementación estándar da interface **TableModel** e derivada de **AbstractTableModel**, que contén os datos da táboa. A propiedade *model* sería posible modificala en calquera momento mediante o método **setModel**. Así poderiamos modificar o código dos apartados anteriores do seguinte xeito:

```
jscrollpane1 = new javax.swing.JScrollPane() ;
taboa1 = new javax.swing.JTable() ;

taboa1.setModel(new javax.swing.table.DefaultTableModel(datos, cabeceiras));
jscrollpane1.setViewportView(taboa1);
```

Agora temos asignado ao obxecto *taboa1* o modelo por defecto **DefaultTableModel**, co cal por empregar o modelo por defecto unha táboa terá as seguintes características:

1. As celdas son todas editables
2. Os datos son todos de tipo String, isto é, dá igual como foran declarados (int, boolean...) a táboa amosaraos como un String; así por exemplo non aparecerá nunha unha casilla de verificación senón o string true ou false.

Para evitar istas restriccións hai que crear un modelo propio creando unha clase que herde de **DefaulTableModel** ou de **AbstractTableModel** e redifinir os métodos necesarios.

Podes atopar máis información sobre modelos de táboas na seguinte ligazón: Taboas personalizadas para interfaces de usuario en Swing

--ricardofc [10/05/10]

1.11 Modelo de eventos swing

Os GUIs son conducidos por eventos; isto significa que responden en distintos xeitos aos distintos eventos que se producen ao longo da súa vida. A maior parte dos eventos son xenerados polo propio usuario. No modelo de eventos hai tres participantes:

- Orixe do evento
- Obxecto do evento
- Listener do evento

A orixe do evento é obxecto que cambia o seu estado e que xenera o evento (por exemplo cando se preme pon unha marca nunha casilla de verificación). O obxecto do evento encapsula o cambio producido no obxecto orixe do evento. O *listener* é o obxecto que quere ser notificado da ocurrencia de ese evento concreto, así o oxecto orixe do evento delega a tarefa de manexar o evento (realizar a acción oportuna) sobre o *listener*. Isto coñécese como o modelo de delegación de eventos de Java. A seguinte táboa mostra unha lista cos listeners(son interfaces) máis usuais.

Interface	Método	Exemplo
ActionListener	actionPerformed()	Click en botóns
AdjustmentListener	adjustmentValueChanged()	Facer scroll
ChangeListener	stateChanged()	Mover un *slider*
ComponentListener	componentResized() etc.	Mover, cambiar tamaño, ocultar ou mostrar
ContainerListener	componentAdded(), componentRemoved()	Engadimos/quitamos un compoñente
FocusListener	focusGained(), focusLost()	Un cambo de texto gaña/perde foco
ItemListener	itemStateChanged()	Cambios nunha *checkbox*
KeyListener	keyPressed(), keyReleased(), keyTyped()	Texto dende o teclado
MouseListener	mouseClicked() etc.	Click do rato
MouseMotionListener	mouseDragged(), mouseMoved()	Todos os movementos do rato
TextListener	textValueChanged()	O contido dun texto cambia
WindowListener	windowActivated() etc.	Pechar unha ventana

Interface Método Exemplo ActionListener actionPerformed() Click en botóns AdjustmentListener adjustmentValueChanged() Facer scroll ChangeListener stateChanged() Mover un 'slider' ComponentListener componentResized() etc. Mover, cambiar tamaño, ocultar ou mostrar ContainerListener componentAdded() componentRemoved() Engadimos/quitamos un compoñente FocusListener focusGained() focusLost() Un cambo de texto gaña/perde foco ItemListener itemStateChanged() Cambios nunha checkbox KeyListener keyPressed() keyReleased() keyTyped() Texto dende o teclado MouseListener mouseClicked() etc. Click do rato MouseMotionListener mouseDragged() mouseMoved() Todos os movementos do rato TextListener textValueChanged() O contido dun texto cambia WindowListener windowActivated() etc. Pechar unha ventana

Cando algo pasa dentro da aplicación créase un obxecto evento. Por exemplo, cando facemos click nun botón ou seleccionamos un elemento de unha lista. Existen moitos tipos de eventos (*ActionEvent, TextEvent, FocusEvent, ComponetEvent*, etc), cada un deles creado baixo determinadas condicións. Examina o seguinte exemplo para ver o funcionamento dos eventos:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.text.DateFormat;

import java.util.Calendar;
import java.util.Date;
import java.util.Locale;

import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JPanel;

public class EventObject extends JFrame {

    private JList list;
    private DefaultListModel model;

    public EventObject() {

        setTitle("Obxecto evento");

        JPanel panel = new JPanel();
        panel.setLayout(null);

        model = new DefaultListModel();
        list = new JList(model);
        list.setBounds(150, 30, 220, 150);

        JButton ok = new JButton("Ok");
        ok.setBounds(30, 35, 80, 25);

        ok.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {

                Calendar cal = Calendar.getInstance();
                cal.setTimeInMillis(event.getWhen());
                Locale locale = Locale.getDefault();
                Date date = new Date();
                String s = DateFormat.getTimeInstance(DateFormat.SHORT,
                    locale).format(date);

                if ( !model.isEmpty() )
                    model.clear();

                if (event.getID() == ActionEvent.ACTION_PERFORMED)
                    model.addElement(" Event Id: ACTION_PERFORMED");

                model.addElement(" Hora: " + s);

                String source = event.getSource().getClass().getName();
                model.addElement(" Fonte: " + source);

                int mod = event.getModifiers();

                StringBuffer buffer = new StringBuffer(" Modificadores: ");

                if ((mod & ActionEvent.ALT_MASK) > 0)
                    buffer.append("Alt ");

                if ((mod & ActionEvent.SHIFT_MASK) > 0)
                    buffer.append("Shift ");

                if ((mod & ActionEvent.META_MASK) > 0)
                    buffer.append("Meta ");

                if ((mod & ActionEvent.CTRL_MASK) > 0)
```

```
                buffer.append("Ctrl ");

            model.addElement(buffer);
        }
    });

    panel.add(ok);
    panel.add(list);
    add(panel);

    setSize(420, 250);
    setLocationRelativeTo(null);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setVisible(true);
  }

  public static void main(String[] args) {
    new EventObject();
  }
}
```

O anterior código mostra un botón e unha lista. Ó pulsar o botón móstrase na lista a información acerca do evento. Neste caso utilizamos o clase *ActionEvent* para capturar o evento clicar nun botón.