Objetivos Interfaces gráficas de usuario y Swing. Trabajo con modelos.

Índice

Como se comentó en las clases de teoría, el tratamiento de eventos en aquellos componentes Swing que utilizan un *model*[1] de tipo datos se debe realizar directamente sobre el *model*.

La clase `JTable` es un ejemplo de este tipo de componente Swing, ya que utiliza un *model* del tipo `TableModel` que es un *model* de datos. Si se desea realizar una tabla en la que se puedan añadir filas o columnas hay que trabajar directamente con el modelo.

La interfaz `TableModel` define una serie de métodos para acceder a los datos contenidos en una determinada posición en la tabla y también para asignar valores a la tabla. A continuación se muestran algunos de los métodos que ofrece esta interfaz (consultad la ayuda para ver la lista completa)

```
1
2  // Para obtener el número de columnas que posee la tabla:
3  public int getColumnCount()
4
5  // Para obtener el número de filas que posee la tabla:
6  public int getRowCount()
7
8  // Para obtener el valor en una determinada celda de la tabla:
9  public Object getValueAt(int rowIndex, int columnIndex)
10
11 // Para establecer el valor de una determinada celda de la tabla:
12 void setValueAt(Object aValue, int rowIndex, int columnIndex)
```

Para realizar un *model* que sustituya al que se ofrece por defecto con la clase `JTable` hay dos opciones:

1. Realizar una clase que implemente todos los métodos de la interfaz `TableModel`.
2. Realizar una clase que extienda a la clase `AbstractTableModel` (Ésta segunda opción es la que se muestra en el código de ejemplo proporcionado). La clase `AbstractTableModel` ya implementa alguno de los métodos de la interfaz `TableModel` pero deja otros 3 métodos de la interfaz sin implementar:

```
1       public int getRowCount();
2       public int getColumnCount();
3       public Object getValueAt(int row, int column);
```

[1] El *model* es el responsable del contenido o estado del componente.

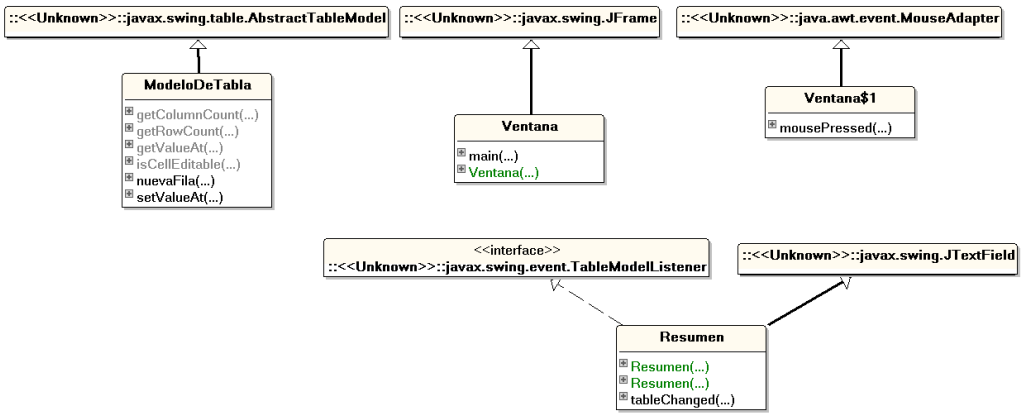

Figura 1: Diagrama de clases del programa de ejemplo.

Aunque lógicamente tenemos la libertad de ocultar el resto si no estamos satisfechos con la implementación ofrecida por la clase `AbstractTableModel`.

Como se ha comentado hay que tratar los eventos que se produzcan sobre el *model*. Una clase que se desee registrar como auditor de eventos del `TableModel` debe implementar a la interfaz `TableModelListener` que tiene un único método:

```
public void tableChanged(TableModelEvent e)
```

1. Un ejemplo

Vamos a ver un ejemplo en el que en la interfaz gráfica de usuario hay una tabla con una columna y un campo de texto en el que se muestra la media de los datos que hay en la tabla. Cuando se modifica el valor de alguna de las celdas o se añaden nuevos datos se actualiza de forma automática la media de los datos.

La figura 1 muestra el diagrama de clases del código de ejemplo y la figura 2 muestra una captura del programa de ejemplo en ejecución junto con las diferentes clases.

```
1
2  import java.awt.*;
3  import java.awt.event.*;
4
5  import javax.swing.*;
6  import javax.swing.event.*;
7  import javax.swing.table.*;
8
9
10 /** La clase Resumen es un campo de texto que implementa a la interfaz TableModelListener
11   * por lo tanto esta clase se puede registrar como oyente de eventos que ocurran en
12   * un objeto del tipo TableModel
13   */
14 class Resumen extends JTextField implements TableModelListener{
15
16    /** Constructor por defecto
```

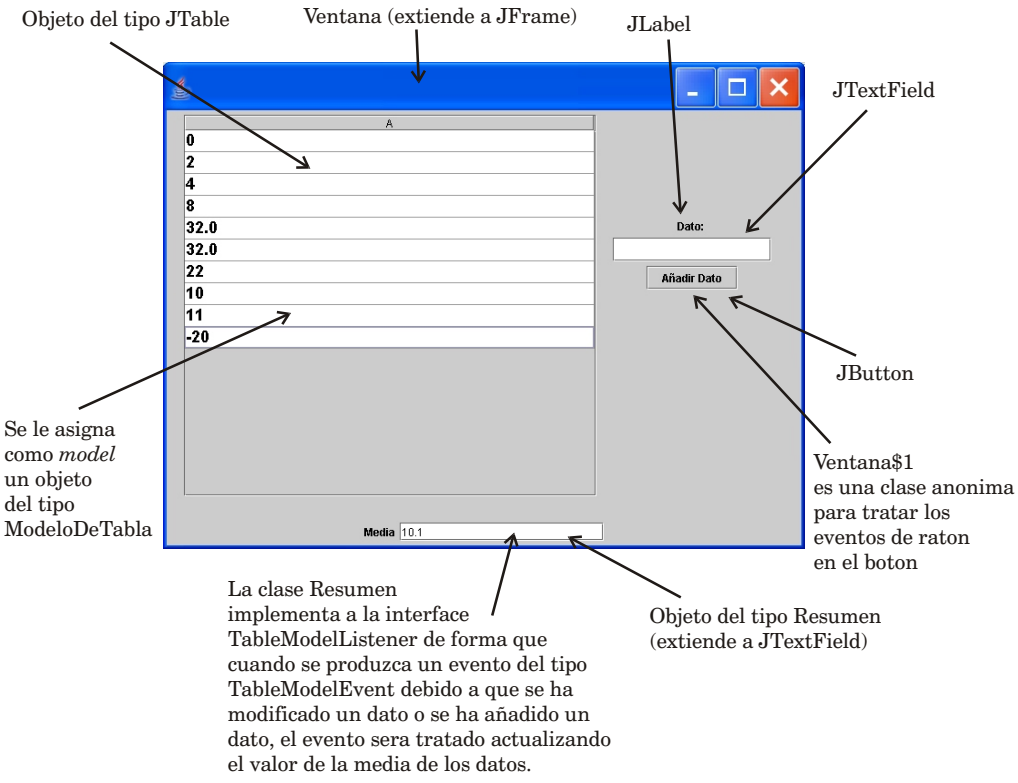

Figura 2: Captura de la pantalla mostrada por el programa de ejemplo mostrando las diferentes clases.

```
17        */
18      public Resumen(){
19          super();
20      }
21
22      /** Constructor en el que se le pasa el tamaño
23        * @param n tamaño en caracteres
24        */
25      public Resumen(int n){
26          super(n);
27      }
28
29      /** Metodo que se define en la interfaz TableModelListener
30        * Obtenemos una referencia al TableModel y obtenemos los datos
31        * que hay en la tabla para recalcular la media y mostrar el resultado
32        * en el campo de texto.
33        * @param TableModelEvent el evento
34        */
35      public void tableChanged(TableModelEvent e) {
36          TableModel tm = (TableModel)e.getSource();
37          double suma = 0;
38          for (int i=0; i<tm.getRowCount();i++)
39              suma = suma + Double.parseDouble((String)tm.getValueAt(i,0));
40
41          setText("" + suma/tm.getRowCount());
42      }
43  }
44
45
46  /** Clase ModeloDeTabla extiende a la clase AbstractTableModel
47    * La clase AbstractTableModel implementa a la interfaz TableModel.
48    * Es mas comodo extender a AbstractTableModel que implementar a TableModel
49    * ya que se deben implementar menos metodos. <br><br>
50    *
51    * Solamente se esta obligado a implementar 3 metodos: <br>
52    *    public int getRowCount(); <br>
53    *    public int getColumnCount(); <br>
54    *    public Object getValueAt(int row, int column); <br>
55    *
56    * Pero se pueden sobreescribir otros en funcion de las necesidades
57    */
58  class ModeloDeTabla extends AbstractTableModel {
59
60      // Datos iniciales
61
62      private String columns[][] =
63          { {"Datos"}};
64      private String rows[][] = {{"0"},
65          {"2"},{"4"},{"8"}};
66
67      /** Metodo definido como abstract en AbstractTableModel
68        * @return el numero de columnas
69        */
70      public int getColumnCount() {
71          return columns.length;
72      }
73
74      /** Metodo definido como abstract en AbstractTableModel
75        * @return el numero de filas
76        */
77      public int getRowCount() {
78          return rows.length;
79      }
80
81      /** Metodo definido como abstract en AbstractTableModel
82        * @return el contenido de una celda en la celda especificada
83        */
84      public Object getValueAt(int row, int column) {
85          return rows[row][column];
86      }
```

```
87
88
89      /** Sobreescribimos el metodo setValueAt de AbstractTableModel
90        * para que se pueda modificar el valor en una celda
91        */
92      public void setValueAt(Object aValue, int row, int column) {
93         rows[row][column] = aValue.toString();
94           // Notificamos a los posibles oyentes que los datos en la tabla han cambiado
95         fireTableDataChanged();
96      }
97
98      /** Sobreescribimos el metodo isCellEditable de AbstractTableModel
99        * para indicar que todas las celdas son editables
100       */
101     public boolean isCellEditable(int row, int column) {
102        return true;
103     }
104
105     /** Metodo nuevo que ofrecemos para añadir una fila a la tabla
106       * @param el valor a introducir
107       */
108     public void nuevaFila(double dx){
109        String[][] datos = new String[getRowCount()+1][getColumnCount()];
110        for (int i=0;i<getRowCount();i++)
111           datos[i] = rows[i];
112         datos[getRowCount()][0] = dx+"";
113
114        rows = datos;
115
116        // Notificamos que los datos en la tabla han cambiado
117        fireTableDataChanged();
118     }
119
120 }
121
122
123 /** Clase que extiende a JFrame
124  * @author Lenguajes de Programacion
125  *
126  * En esta clase se construye la interfaz gráfica de usuario.
127  * Se crea una tabla (JTable) y se le asigna como modelo un objeto de la
128  * clase anterior (ModeloDeTabla).
129  *
130  */
131 public class Ventana extends JFrame{
132
133    public Ventana(){
134       Container cp = getContentPane();
135       cp.setLayout(new BorderLayout());
136
137       final JTable tabla = new JTable();
138
139       // Le asignamos como modelo un objeto de la clase ModeloDeTabla
140       // De esta forma controlamos como se debe comportar la tabla.
141       tabla.setModel(new ModeloDeTabla());
142
143       tabla.setFont(new Font("Arial",Font.BOLD,18));
144       tabla.setRowHeight(24);
145
146
147       // Esto lo hacemos para que el dato que se esta editando aparezca
148       // resaltado en otro color
149       JTextField f = new JTextField();
150       f.setForeground(Color.red);
151       f.setFont(new Font("Arial",Font.PLAIN,18));
152       tabla.getColumnModel().getColumn(0).setCellEditor(new DefaultCellEditor(f));
153
154       // Ponemos la tabla dentro de un JScrollPane
155       JScrollPane jsp = new JScrollPane(tabla);
156
```

```
157
158         // Vamos a organizar los componentes dentro de varios contenedores
159
160         JPanel p11 = new JPanel();
161
162         p11.setPreferredSize(new Dimension(200,200));
163
164         p11.add(new JLabel("Dato: "));
165
166         final JTextField dato = new JTextField(10);
167         dato.setFont(new Font("Arial",Font.BOLD,18));
168
169         p11.add(dato);
170
171         final JButton b1 = new JButton("Añadir Dato");
172
173
174         p11.add(b1);
175
176         // Registramos un objeto oyente de eventos de raton.
177         // Si se pulsa en boton añadimos una fila mas en la tabla con
178         // el dato que se haya introducido en el campo de texto.
179         // Si no es un valor numerico, se muestra un mensaje de error
180         // (utilizando la clase JOptionPane).
181         b1.addMouseListener(new MouseAdapter(){
182             public void mousePressed(MouseEvent e){
183                 ModeloDeTabla tm = (ModeloDeTabla)tabla.getModel();
184                 try{
185                     double valor = Double.parseDouble(dato.getText());
186                     tm.nuevaFila(valor);
187                     dato.setText("");
188                 }catch(NumberFormatException ex){
189                 JOptionPane.showMessageDialog(
190                     null,
191                     "El dato introducido no es un número",
192                     "Atención",
193                     JOptionPane.INFORMATION_MESSAGE);
194
195               }
196             }
197         });
198
199         JPanel p1 = new JPanel();
200
201         p1.add(jsp);
202
203         p1.add(p11);
204
205
206         Resumen r = new Resumen(20);
207
208         tabla.getModel().addTableModelListener(r);
209
210         Panel p2 = new Panel();
211         p2.setLayout(new FlowLayout());
212         p2.add(new JLabel("Media"));
213
214         p2.add(r);
215
216         cp.add(p1,BorderLayout.CENTER);
217         cp.add(p2,BorderLayout.SOUTH);
218
219         // Otra forma de decir que si se cierra la ventana finalice la aplicacion
220         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
221
222         setSize(900,700);
223         setVisible(true);
224     }
225
226     public static void main(String[] args){
```

```
227         new Ventana();
228     }
229
230 }
```

Como se puede observar en el código, la clase `Resumen` (linea 14) extiende a la clase `JTextField` e implementa a la interfaz `TableModelListener`. De este modo es posible registrar (linea 202) un objeto de este tipo como oyente de eventos del tipo `TableModelEvent` que se produzcan en la tabla.

Al modificar el valor de una celda (linea 93) o insertar una nueva fila (linea 115) en la tabla se fuerza a que se dispare un evento.

Este evento será capturado por el objeto del tipo `Resumen` que actualiza la media (linea 33).

2. Tareas

Se pide modificar el programa anterior de forma que en lugar de aparecer una tabla con datos iniciales fijados de antemano, los datos se obtengan desde un fichero de texto. El nombre del fichero se debe leer desde un campo de texto, siendo posible cambiar el nombre del mismo en cualquier momento con lo cual la tabla deberá mostrar los nuevos datos. Los datos contenidos en la tabla se deben poder almacenar en un fichero.

Todos los datos necesarios para el funcionamiento de la aplicación se deben obtener siempre desde la interfaz gráfica de usuario.

Se debe permitir añadir nuevos datos (como en el código de ejemplo).

Además, en lugar de mostrar únicamente la media en un campo de texto como en el código de ejemplo, hay que mostrar la media, la varianza, el mínimo, el máximo y la mediana para lo que se deberá utilizar un área de texto en lugar de un campo de texto.

Se proporcionan los siguientes ficheros:

- Código fuente de ejemplo
- Un fichero de datos correspondientes a la evolución del EURIBOR a 12 meses[2] entre el 3 de enero de 2004 y el 20 de abril de 2005 (335 valores).

[2]Estos datos son publicados por el Banco de España (`http://www.bde.es`)