



Tema 5

Componentes swing

Proyecto de Análisis y Diseño de Software
2º Ingeniería Informática
Universidad Autónoma de Madrid

Password: JPasswordField

Campo de texto donde los caracteres tecleados se muestran como *.

```
// Crear el campo de tipo password. El argumento del constructor indica la
// anchura del campo en pantalla, no el número de caracteres de la password.
JPasswordField campo = new JPasswordField(10);

// Para obtener el valor del campo usar el método getPassword,
// que devuelve un array de caracteres
char[] password = campo.getPassword();

// Añadir el campo al panel donde se quiera mostrar
JPanel ejemploPasswordField = new JPanel(new GridLayout(2,1,2,2));
ejemploPasswordField.add(new JLabel("Introduce la password"));
ejemploPasswordField.add(campo);
```



Casillas de verificación: JCheckBox

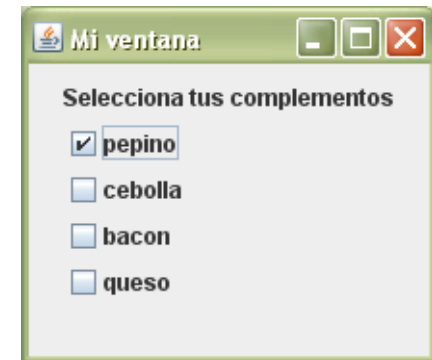
Cada casilla puede estar seleccionada o no.

```
// Crear las opciones
JCheckBox casilla1 = new JCheckBox("pepino");
JCheckBox casilla2 = new JCheckBox("cebolla");
JCheckBox casilla3 = new JCheckBox("bacon");
JCheckBox casilla4 = new JCheckBox("queso");

// Para consultar si una casilla está seleccionada,
// usar el método isSelected
if (casilla1.isSelected())
    System.out.println("La casilla 1 está seleccionada");

// Para cambiar el valor de una casilla, usar el método setSelected
casilla1.setSelected(true);

// Añadir las casillas al panel donde se quieren mostrar
JPanel ejemploCheckbox = new JPanel(new GridLayout(5,1));
ejemploCheckbox.add(new JLabel("Selecciona tus complementos"));
ejemploCheckbox.add(casilla1);
ejemploCheckbox.add(casilla2);
ejemploCheckbox.add(casilla3);
ejemploCheckbox.add(casilla4);
```



Botones de radio: JRadioButton

En un grupo sólo una de las opciones está seleccionada.

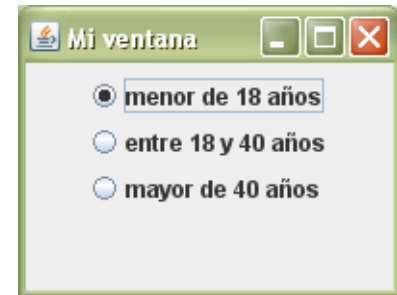
```
// Crear las opciones
JRadioButton opcion1 = new JRadioButton("menor de 18 años");
JRadioButton opcion2 = new JRadioButton("entre 18 y 40 años");
JRadioButton opcion3 = new JRadioButton("mayor de 40 años");
opcion1.setSelected(true);

// Crear un grupo para las opciones, el cual garantizará
// que sólo una de las opciones está seleccionada
ButtonGroup grupo = new ButtonGroup();

// Añadir las opciones al grupo
grupo.add(opcion1);
grupo.add(opcion2);
grupo.add(opcion3);

// Añadir las opciones al panel donde se quieran mostrar
JPanel ejemploRadioButton = new JPanel(new GridLayout(3, 1));
ejemploRadioButton.add(opcion1);
ejemploRadioButton.add(opcion2);
ejemploRadioButton.add(opcion3);

// Para consultar si una opción está seleccionada, usar el método isSelected
if (opcion1.isSelected())
    System.out.println("Has seleccionado menor de 18 años");
```

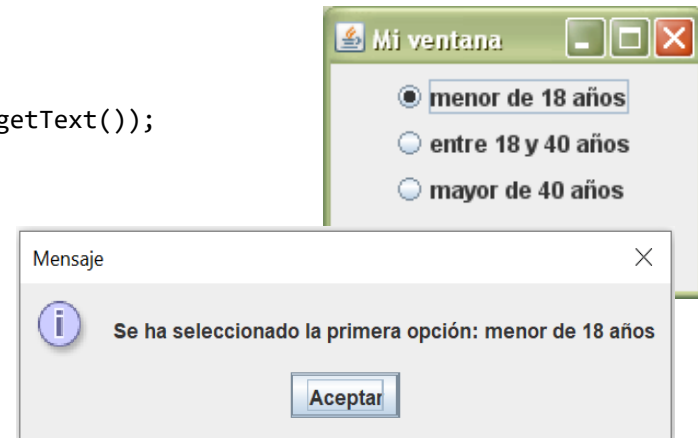


Botones de radio: JRadioButton

Reacción a eventos de selección

```
// ... continuación del ejemplo anterior
```

```
ActionListener listener = new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        if (e.getSource() == opcion1)  
            JOptionPane.showMessageDialog(null,  
                "Se ha seleccionado la primera opción: "+opcion1.getText());  
    }  
};  
opcion1.addActionListener(listener);  
opcion2.addActionListener(listener);  
opcion3.addActionListener(listener);
```



Combos: JComboBox

Permite seleccionar una opción dentro de una lista desplegable.

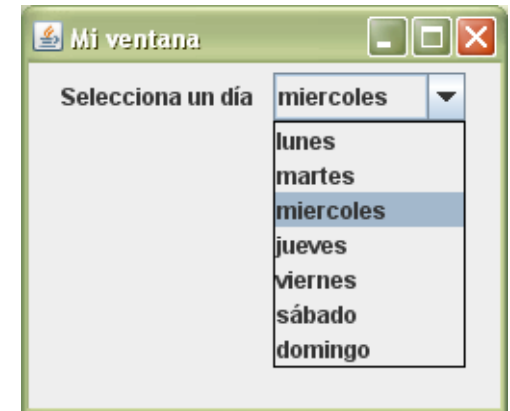
```
// Crear un array con las opciones del combo
String[] dias = {"lunes", "martes", "miercoles", "jueves", "viernes", "sábado", "domingo"};

// Crear el combo, pasando el array como parámetro
JComboBox<String> combo = new JComboBox<String>(dias);

// Para seleccionar por código una opción, usar setSelectedIndex(<posicion>)
combo.setSelectedIndex(2);

// Añadir el combo al panel donde se quiera mostrar
JPanel ejemploComboBox = new JPanel(new GridLayout(1,2,10,0));
ejemploComboBox.add(new JLabel("Selecciona un día"));
ejemploComboBox.add(combo);

// Para saber qué entrada está seleccionada,
// usar getSelectedItem o setSelectedIndex
String valorSeleccionado = (String)combo.getSelectedItem();
int indiceSeleccionado = combo.getSelectedIndex();
```



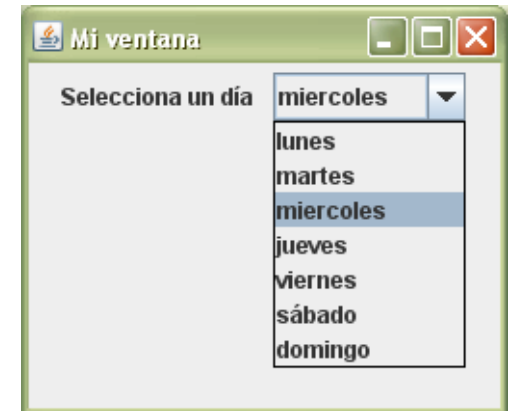
Combos: JComboBox

Reacción a eventos de selección

```
// ... continuación del ejemplo anterior
```

```
combo.addItemListener(new ItemListener() {  
    @Override  
    public void itemStateChanged(ItemEvent e) {  
        int state = e.getStateChange();  
        if (state==ItemEvent.SELECTED)  
            System.out.println("Selected: " + e.getItem().toString());  
        else System.out.println("Deselected: " + e.getItem().toString());  
    }  
});
```

```
Deselected: miercoles  
Selected:    viernes
```



Listas: JList (lista inmutable)

En este primer ejemplo, al crear la lista se le pasan sus elementos, y después ya **no** es posible añadir o eliminar elementos de la lista.

```
// Crear un array con los elementos de la lista
String[] personas = {"ana", "eduardo", "esther", "josé", "juan", "luis", "maría", "miguel", "zoe"};

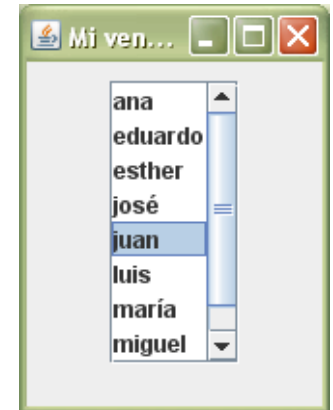
// Crear la lista, pasando el array como parámetro
JList<String> lista = new JList<String>(personas);

// Por defecto se pueden seleccionar varias filas en la lista. Si se
// quiere que sólo pueda seleccionarse una, usar setSelectionMode
lista.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

// Es aconsejable crear una barra de scroll para la lista,
// por si el número de elementos supera el tamaño previsto
JScrollPane scroll = new JScrollPane(lista);

// Añadir el scroll con la lista al panel donde se vaya a mostrar
JPanel ejemploList = new JPanel();
ejemploList.add(scroll);

// Para realizar una acción al seleccionar/deseleccionar una entrada, usar un listener
lista.addListSelectionListener(new ListSelectionListener () {
    public void valueChanged(ListSelectionEvent arg0) {
        if (arg0.getValueIsAdjusting()==false) {
            // Obtener una referencia a la lista que ha cambiado
            JList<String> lista = (JList<String>)arg0.getSource();
            // Para saber qué entrada está seleccionada, usar getSelectedValue o getSelectedIndex
            String valorSeleccionado = (String)lista.getSelectedValue();
            int indiceSeleccionado = lista.getSelectedIndex();
        }
    }
});
```



Listas: JList (lista dinámica)

Si se quiere añadir y eliminar elementos a la lista dinámicamente después de haberla creado, hay que usar un DefaultListModel que contenga los datos.

```
// Crear un DefaultListModel con los elementos de la lista
DefaultListModel<String> personasM = new DefaultListModel<String>();
personasM.addElement("ana");
personasM.addElement("eduardo");
personasM.addElement("esther");
personasM.addElement("josé");
personasM.addElement("juan");
personasM.addElement("luis");
personasM.addElement("maría");
personasM.addElement("miguel");
personasM.addElement("zoe");

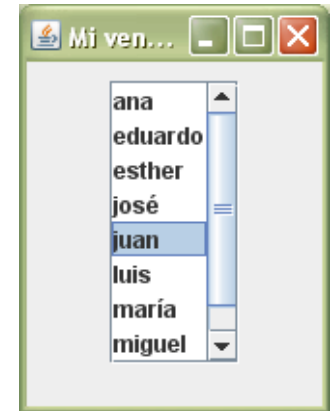
// Crear la lista, pasando el modelo como parámetro
JList<String> lista = new JList<String>(personasM);

// Para eliminar un elemento de la lista llamamos al método
// remove(<index>) o remove(<object>) del modelo
personasM.remove(0); // borra ana
personasM.removeElement("esther"); // borra esther

// Para añadir un elemento al final de la lista, llamamos al método addElement del modelo
personasM.addElement("perico");

// También podemos insertar un elemento en una posición, con el método add(<index>,<object>) del modelo
personasM.add(0, "lucas");

// Para modificar un elemento existente, usamos el método set(<index>,<object>) del modelo
personasM.set(1, "belén"); // cambia eduardo por belén
```



Tablas: JTable (tabla inmutable)

En este primer ejemplo, al crear la tabla se le pasan sus filas, y después ya no es posible añadir o quitar filas. Las celdas de la tabla son editables, y muestran el “*toString*” del objeto que contienen.

```
// Crear un array con el título de las columnas
String[] titulos = {"nombre", "apellidos", "nota", "aprobado"};
```

```
// Crear una matriz con las filas de la tabla
Object[][] filas = {
    {"Alba", "Sanz", new Double(5), new Boolean(true)},
    {"Belén", "López", new Double(3.2), new Boolean(false)},
    {"Luisa", "López", new Double(4.5), new Boolean(false)},
    {"Marcos", "Pérez", new Double(8.5), new Boolean(true)},
    {"Miguel", "Vela", new Double(7), new Boolean(true)},
    {"Sara", "Valle", new Double(10), new Boolean(true)},
};
```

```
// Crear la tabla, pasando las filas y los títulos como parámetro
JTable tabla = new JTable(filas, titulos);
```

```
// Por defecto se pueden seleccionar varias filas en la tabla. Si se
// quiere que sólo pueda seleccionarse una, usar setSelectionMode
tabla.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

```
// Podemos fijar el tamaño de la tabla
tabla.setPreferredScrollableViewportSize(new Dimension(500, 80));
```

```
// Es aconsejable crear una barra de scroll para la tabla, por si el número de filas supera el tamaño previsto
JScrollPane scroll = new JScrollPane(tabla);
```

```
// Añadir el scroll con la tabla al panel donde se vaya a mostrar
JPanel ejemploTable = new JPanel();
ejemploTable.add(scroll);
```



nombre	apellidos	nota	aprobado
Alba	Sanz	5.0	true
Belén	López	3.2	false
Luisa	López	4.5	false
Marcos	Pérez	8.5	true
Miguel	Vela	7.0	true
Miguel	Vela	7.0	true

Tablas: JTable (tabla dinámica)

Si se quiere añadir y eliminar filas a la tabla dinámicamente después de haberla creado, hay que usar un **DefaultTableModel** que contenga los datos.

```
// Crear un array con el título de las columnas
String[] titulos = {"nombre", "apellidos", "nota", "aprobado"};
```

```
// Crear una matriz con las filas de la tabla
Object[][] filas = {
    {"Alba", "Sanz", new Double(5), new Boolean(true)},
    {"Belén", "López", new Double(3.2), new Boolean(false)},
    {"Luisa", "López", new Double(4.5), new Boolean(false)},
    {"Marcos", "Pérez", new Double(8.5), new Boolean(true)},
    {"Miguel", "Vela", new Double(7), new Boolean(true)},
    {"Sara", "Valle", new Double(10), new Boolean(true)},
};
```

```
// Crear un DefaultTableModel con las filas y los títulos de la tabla
DefaultTableModel modeloDatos = new DefaultTableModel(filas, titulos);
```

```
// Crear la tabla, pasando el modelo como parámetro
JTable tabla = new JTable(modeloDatos);
```

```
// Para eliminar un elemento de la tabla llamamos al método removeRow(<index>) del modelo
modeloDatos.removeRow(0); // borra alba
```

```
// Para añadir un elemento al final de la tabla, llamamos al método addRow del modelo
Object[] nuevaFila = {"Victor", "Téllez", new Double(6), new Boolean(true)};
modeloDatos.addRow(nuevaFila);
```

```
// También podemos insertar una fila en una posición, con el método insertRow(<index>,<object[]>) del modelo
modeloDatos.insertRow(0, nuevaFila);
```

```
// Para modificar una fila existente, usamos el método set(<object>,<fila>,<columna>) del modelo
modeloDatos.setValueAt("Juan", 1, 0); // cambia belén por juan
```

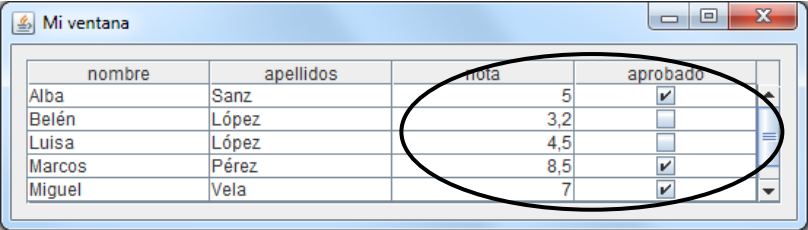


nombre	apellidos	nota	aprobado
Víctor	Téllez	6.0	true
Juan	López	3.2	false
Luisa	López	4.5	false
Marcos	Pérez	8.5	true
Miguel	Vela	7.0	true

Tablas: JTable (tabla personalizada)

Para controlar cómo se muestran los datos en la tabla, hay que crear un modelo de tabla personalizado (clase que extiende **AbstractTableModel**).

```
public class MiModeloPersonalizado extends AbstractTableModel {  
  
    // La clase debe guardar como atributos los datos de la tabla.  
    private String[] titulos;  
    private Object[][] filas;  
  
    // En el constructor se dará valor a dichos atributos.  
    public MiModeloPersonalizado () { ... }  
  
    // La clase debe implementar obligatoriamente los métodos getColumnCount, getRowCount y getValueAt  
    public int getColumnCount() { return titulos.length; } // Método que devuelve el número de columnas  
    public int getRowCount() { return filas.length; } // Método que devuelve el número de filas  
    public Object getValueAt(int row, int col) { return filas[row][col]; } // Método que devuelve el contenido de una celda  
  
    // Por defecto las celdas de la tabla no podrán editarse. Para hacerlas editables sobrescribir isCellEditable  
    public boolean isCellEditable (int row, int col) { return row==0; } // permite editar la primera fila  
  
    // Si se quiere cambiar el valor de las celdas una vez creadas, sobrescribir setValueAt  
    public void setValueAt (Object value, int row, int col) {  
        filas[row][col] = value;  
        fireTableCellUpdated(row, col);  
    }  
  
    // Podemos sobrescribir getColumnNames para devolver el título de una columna  
    public String getColumnNames (int col) { return titulos[col]; }  
  
    // Podemos visualizar los datos de las celdas con el formato predefinido sobrescribiendo getColumnClass.  
    // Los boolean se mostrarán como un checkbox, y los números estarán alineados a la derecha (ver figura).  
    public Class<?> getColumnClass (int col) { return getValueAt(0, col).getClass(); }  
  
}
```



nombre	apellidos	nota	aprobado
Alba	Sanz	5	<input checked="" type="checkbox"/>
Belén	López	3,2	<input type="checkbox"/>
Luisa	López	4,5	<input type="checkbox"/>
Marcos	Pérez	8,5	<input checked="" type="checkbox"/>
Miguel	Vela	7	<input checked="" type="checkbox"/>

Tablas: JTable (tabla personalizada)

Una vez creado el modelo de tabla personalizado podemos instanciarlo así:

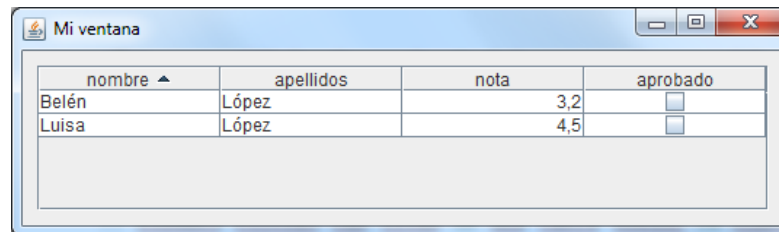
```
// Crear un modelos de datos
AbstractTableModel modeloDatos = new MiModeloPersonalizado();

// Crear la tabla, pasando el modelo como parámetro
JTable tabla = new JTable(modeloDatos);
```

Otras utilidades de JTable: ordenación y filtros

```
// El método setAutoCreateRowSorter permite ordenar una tabla al pulsar la cabecera de sus columnas
tabla.setAutoCreateRowSorter(true);
```

```
// Podemos filtrar las filas de una tabla usando la clase TableRowSorter. El siguiente ejemplo
// filtra la tabla para que sólo aparezcan las filas en las que alguna celda contiene "López"
TableRowSorter<MiModeloPersonalizado> filtro = new TableRowSorter(modeloDatos);
RowFilter<MiModeloPersonalizado, Integer> rf = RowFilter.regexFilter("López");
filtro.setRowFilter(rf);
tabla.setRowSorter(filtro);
```



nombre ▲	apellidos	nota	aprobado
Belén	López	3,2	<input type="checkbox"/>
Luisa	López	4,5	<input type="checkbox"/>

Árboles: JTree

Proporciona una vista jerárquica de un conjunto de datos

```
// Crear el nodo raíz del árbol, pasando el texto que mostrará
DefaultMutableTreeNode raiz = new DefaultMutableTreeNode("Apuntes de PADS");

// Crear el árbol, pasándole el nodo raíz
JTree arbol = new JTree (raiz);

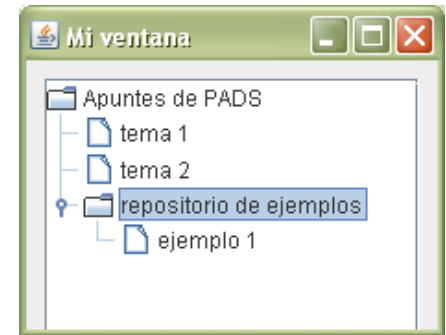
// Por defecto se pueden seleccionar varios nodos en el árbol. Para poder seleccionar sólo uno, usar setSelectionModel
arbol.getSelectionModel().setSelectionMode(TreeSelectionModel.SINGLE_TREE_SELECTION);

// Para añadir hijos al nodo raíz (o a otros nodos del árbol) usamos el método add(<nodo>)
raiz.add(new DefaultMutableTreeNode("tema 1"));
raiz.add(new DefaultMutableTreeNode("tema 2"));
DefaultMutableTreeNode repositorio = new DefaultMutableTreeNode("repositorio de ejemplos");
repositorio.add(new DefaultMutableTreeNode("ejemplo 1"));
raiz.add(repositorio);

// Para realizar acciones al seleccionar un nodo del árbol usamos un TreeSelectionListener
arbol.addTreeSelectionListener(new TreeSelectionListener() {
    public void valueChanged(TreeSelectionEvent e) {
        // Para obtener el nodo seleccionado en el árbol:
        // - si el modo de selección es SINGLE_TREE_SELECTION, usamos getLastSelectedPathComponent
        Object nodo = arbol.getLastSelectedPathComponent();
        // - si se pueden seleccionar varios nodos, usamos getSelectionPaths / getSelectionRows
        int[] indiceNodosSeleccionados = arbol.getSelectionRows();
        TreePath[] pathNodosSeleccionados = arbol.getSelectionPaths();
    }
});

// Es aconsejable crear una barra de scroll para el árbol, por si se supera el tamaño previsto
JScrollPane scroll = new JScrollPane(arbol);

// Añadir el scroll con la tabla al panel donde se vaya a mostrar
JPanel ejemploTree = new JPanel();
ejemploTree.add(scroll);
```



Árboles: JTree (árbol dinámico)

Si se quiere añadir y eliminar nodos al árbol dinámicamente, hay que usar un **DefaultTreeModel** que contenga los datos.

```
// Crear el nodo raíz del árbol, pasando el texto que mostrará
DefaultMutableTreeNode raiz = new DefaultMutableTreeNode("Apuntes de PADS");

// Crear el modelo de datos del árbol, pasando el nodo raíz
DefaultTreeModel modeloDatos = new DefaultTreeModel(raiz);

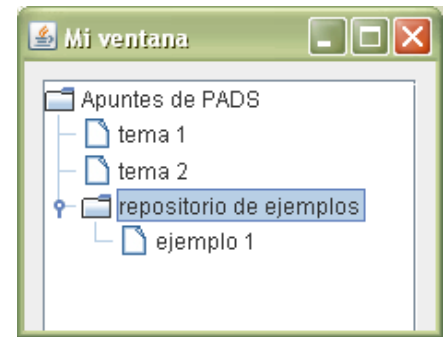
// Crear el árbol, pasándole el modelo de datos
JTree arbol = new JTree(modeloDatos);

// Podemos fijar el tamaño del árbol
arbol.setPreferredSize(new Dimension(200, 40));

// Para añadir hijos a un nodo usamos el método insertNodeInto del modelo. El método recibe el nodo
// a insertar, el nodo padre donde se inserta, y la posición del nodo entre los hijos del padre.
modeloDatos.insertNodeInto(new DefaultMutableTreeNode("tema 1"), raiz, 0); // "tema 1" es primer hijo de raíz
modeloDatos.insertNodeInto(new DefaultMutableTreeNode("tema 2"), raiz, 1); // "tema 2" es segundo hijo de raíz
DefaultMutableTreeNode repositorio = new DefaultMutableTreeNode("repositorio de ejemplos");
modeloDatos.insertNodeInto(repositorio, raiz, 2); // "repositorio..." es tercer hijo de raíz
modeloDatos.insertNodeInto(new DefaultMutableTreeNode("ejemplo 1"), repositorio, 0); // "ejemplo 1" es primer
// hijo de "repositorio..."

// Para eliminar un nodo usamos removeFromParent(<nodo>). También se borrarán sus hijos.
modeloDatos.removeNodeFromParent(repositorio);

// Para modificar el valor del nodo seleccionado:
if (!arbol.isSelectionEmpty()) {
    TreePath path = arbol.getSelectionPath(); // obtener path al nodo seleccionado
    modeloDatos.valueForPathChanged(path, "tema 20"); // asignarle el valor nuevo
}
```



Pestañas: JTabbedPane

Permite organizar las distintas “pantallas” como pestañas.

JTabbedPane es un contenedor (igual que JPanel), no un componente gráfico.

```
// Creamos un panel por cada pestaña
JPanel pestania1 = new JPanel();
pestania1.setLayout(new FlowLayout());
pestania1.add(new JLabel("Nombre"));
pestania1.add(new JTextField(10));
pestania1.add(new JButton("Aceptar"));
pestania1.setPreferredSize(new Dimension(300, 100));

// Otros dos paneles
JPanel pestania2 = new JPanel();
JPanel pestania3 = new JPanel();

// Creamos un contenedor de tipo JTabbedPane (no JPanel).
JTabbedPane pestanias = new JTabbedPane();

// Añadimos los paneles al contenedor con el método addTab(<título>,<panel>)
pestanias.addTab("Pestaña 1", pestania1);
pestanias.addTab("Pestaña 2", pestania2);
// El método addTab también permite asociar un icono y ayuda contextual a la pestaña
pestanias.addTab("Pestaña 3", new ImageIcon("Image3.gif"), pestania3, "Esta pestaña hace cosas");

// Podemos seleccionar una pestaña del contenedor con setSelectedIndex(<índice>)
pestanias.setSelectedIndex(2);

// Podemos eliminar una pestaña del contenedor con removeTabAt(<índice>)
pestanias.removeTabAt(1);

// Para realizar acciones al cambiar de pestañas definiremos un ChangeListener
pestanias.addChangeListener(new ChangeListener() {
    public void stateChanged(ChangeEvent e) { ... }
});
```

