Componentes visuales III

Menús, JFileChooser, JTable

Tecnología de la Programación

Curso 2019-2020

Jesús Correas – jcorreas@ucm.es

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

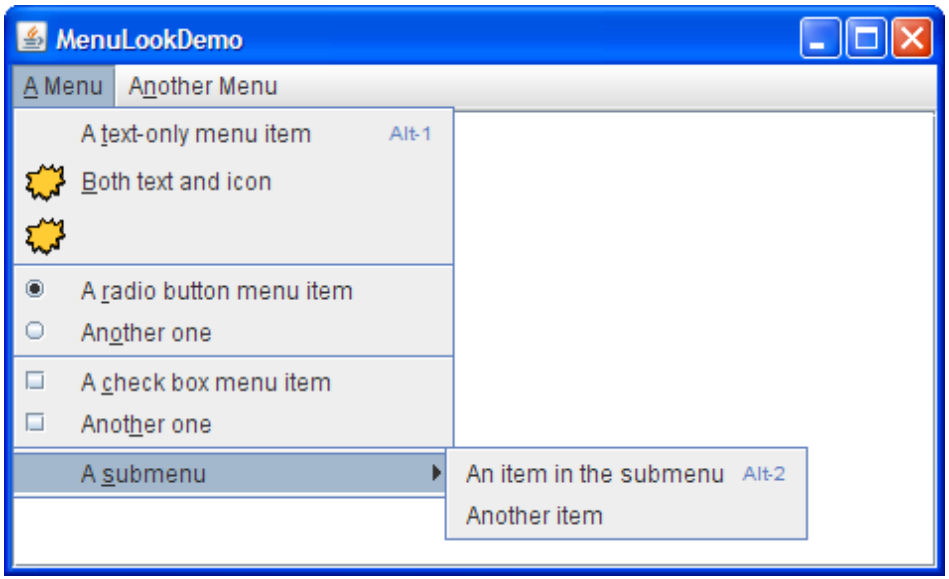
(Basado en material creado por Yolanda García y en `http://docs.oracle.com/javase/tutorial/uiswing/components`)

Menús

- En casi todas las aplicaciones de ventana se utilizan dos tipos de menú principales:
  - La **barra de menús** que se muestra en la ventana principal de la mayor parte de las aplicaciones.
  - Los **menús desplegables** (*pop-up*) que aparecen cuando se pulsa con el botón derecho del ratón en determinadas zonas de la ventana.
- En ambos casos los elementos utilizados son muy parecidos: una lista de entradas en las que se puede seleccionar una para ejecutar una acción.

Barra de menús

- Primero vamos a ver cómo programar una **barra de menús** en una ventana. Para ello se utiliza un objeto de tipo **`JMenuBar`**.



- En una barra de menú tenemos una **lista de menús**.
- En cada menú tenemos una **lista de opciones de menú**.
  - Cada opción de menú puede tener a su vez un submenú.
  - Las opciones de menú se pueden comportar como *check boxes* o *radio buttons*.

(Imagen: `http://docs.oracle.com/javase/tutorial/figures/uiswing/components/MenuLookDemo.png`)

Barra de menús

- El objeto **`JMenuBar`** permite definir la barra de menú que se asocia a la ventana. Para ello se utiliza el método **`setJMenuBar`** de la ventana (`JFrame`).
- A la barra de menú se le pueden añadir los menús, objetos de tipo **`JMenu`**.
- A cada menú, a su vez, se le pueden añadir **opciones de menú** **`JMenuItem`** o submenús (otros objetos **`JMenu`**).
- También se pueden añadir **separadores** **`JSeparator`**, líneas que separan opciones de un menú.
- A las opciones de menú se pueden asociar manejadores **`ActionListener`**.
- **`EjemploJMenuBar1.java`**.

Variantes en opciones de Menú

- Las opciones de menú (**`JMenuItem`**) en realidad son similares a botones.
- Como en el caso de los botones, existen opciones de menú que son *check boxes* (**`JCheckBoxMenuItem`**) o *radio buttons* (**`JRadioButtonMenuItem`**).
- Funcionan de forma similar a los botones de los tipos correspondientes:
- Mantienen información del estado: si la opción está seleccionada o no.
- Tienen eventos característicos.
- Las opciones de menú de tipo `JRadioButton` deben agruparse para mantener la selección de un único botón en el grupo.
- Esto permite tener varios grupos de opciones de menú de este tipo.
- **`EjemploJMenuBar2.java`**.

Menús desplegables (popup)

- También se pueden asociar menús emergentes cuando se pulsa sobre determinados componentes del entorno de ventana.
- En Java se debe mostrar manualmente el menú emergente en el evento del ratón que se desee.
- Por ejemplo, se puede asociar al evento de pulsar con el botón derecho del ratón: el evento **`MouseEvent`**, que se añade al componente que se desee con el método **`addMouseListener`**.
- Cuando se crea un manejador de eventos del ratón, No se suele crear una clase que implementa **`MouseListener`**.
- `MouseListener` contiene varios métodos para manejar **todo** el comportamiento del ratón:
  - `mouseClicked(MouseEvent e)`
  - `mouseEntered(MouseEvent e)`
  - `mouseExited(MouseEvent e)`
  - `mousePressed(MouseEvent e)`
  - `mouseReleased(MouseEvent e)`

Menús desplegables (popup)

- En su lugar, se suele **crear una clase que hereda de `MouseAdapter`**
- **`MouseAdapter`** contiene una implementación vacía de todos los métodos de `MouseListener`.
  - Así, si queremos implementar una sola acción del ratón, basta con sobrescribir el método de `MouseAdapter` que nos interese.
  - Por ejemplo, cuando se pulsa con el ratón sobre el componente: **`mousePressed(MouseEvent e)`**.
- Pero un menú emergente aparece de forma distinta según el sistema en el que nos encontremos (por ejemplo, pulsando el botón derecho, etc.). Es el *look & feel* del sistema.
  - Podemos saber si se ha pulsado el ratón para mostrar un menú pop-up utilizando **`isPopupTrigger()`** **sobre el objeto de tipo `MouseEvent`**.
- **`EjemploJPopupMenu1.java`**.

Combinaciones de teclas (mnemonics) y aceleradores

- Son procedimientos para poder acceder a una opción de menú utilizando el teclado.
- Las combinaciones de teclas son normalmente letras del título del menú u opción de menú que pulsándolas permiten acceder a la opción cuando el menú está desplegado.
- Se puede fijar la combinación de teclas de una opción o menú con **`setMnemonic`**, o bien en la constructora del objeto `JMenuItem` o `JMenu`.
- Los aceleradores son accesos rápidos a una opción de menú aunque esta opción no esté visible.
- En este caso se utiliza **`setAccelerator`**.
- En ambos casos se debe pasar la combinación de teclas utilizada, que es un número entero (hay constantes definidas en **`KeyEvent`**) o bien un objeto **`KeyStroke`**.
- **`EjemploJMenuBar3.java`**.
- Más información:

`http://docs.oracle.com/javase/tutorial/uiswing/components/menu.html`

Otros cuadros de diálogo: `JFileChooser`

Más cuadros de diálogo

- Además de los cuadros de diálogo sencillos que hemos visto en la clase **`JOptionPane`**, existen otros:
  - **`JColorChooser`** para seleccionar un color de una paleta de colores.
  - **`JFileChooser`** para seleccionar un nombre de archivo del sistema de ficheros.
  - Cuadros de diálogo de **configuración de impresión** y de **formato de página**.
- Estos cuadros de diálogo estándar realizan tareas comunes a muchas aplicaciones y evitan tener que programarlos en cada aplicación.
- Nos centraremos en la clase **`JFileChooser`**.

Clase `JFileChooser`

- Esta clase nos permite navegar por el sistema de ficheros para seleccionar un nombre de fichero.
- **No abre ningún fichero.** Contiene métodos para mostrar un cuadro de diálogo **modal** que permite **seleccionar un nombre de fichero.**
- El funcionamiento habitual de este componente es el siguiente:
  1. Se crea un objeto de tipo `JFileChooser`.
  2. Se configuran las opciones del cuadro de diálogo: seleccionar los tipos de ficheros que se deben mostrar, el directorio inicial, etc.
  3. Se muestra el cuadro de diálogo.
  4. Se obtiene el nombre del fichero que ha seleccionado el usuario.

Cuadros de diálogo de `JFileChooser`

- Se pueden configurar distintos aspectos del cuadro de diálogo:
- El **tipo de ficheros que se muestran**. Por defecto, solo se pueden seleccionar ficheros, pero se puede modificar con **`setFileSelectionMode`** para seleccionar directorios o ambos con las siguientes constantes:
  - **`JFileChooser.DIRECTORIES_ONLY`**
  - **`JFileChooser.FILES_ONLY`**
  - **`JFileChooser.FILES_AND_DIRECTORIES`**
- **`setMultiSelectionEnabled`** para seleccionar múltiples archivos.
- **`setFileFilter`** para establecer un filtro sobre los archivos del sistema de ficheros: por ejemplo, el objeto **`FileNameExtensionFilter`** permite filtrar archivos por extensión.
- **`setCurrentDirectory`** para establecer el directorio inicial del cuadro de diálogo.

Cuadros de diálogo de `JFileChooser`

- Los métodos para mostrar los cuadros de diálogo son:
  - **`showOpenDialog(Component parent)`** Muestra un cuadro de diálogo para **abrir un fichero.**
  - **`showSaveDialog(Component parent)`** Muestra un cuadro de diálogo para **guardar un fichero.**
  - **`showDialog(Component parent, String approveButtonText)`** muestra un cuadro de diálogo en el que podemos fijar el texto del botón "aceptar".
- El valor de retorno de estos métodos es un número entero que nos permite distinguir cómo ha cerrado el usuario el cuadro de diálogo:
  - **`JFileChooser.APPROVE_OPTION`** indica que el usuario pulsó **aceptar.**
  - **`JFileChooser.CANCEL_OPTION`** indica que se pulsó **cancelar.**
  - **`JFileChooser.ERROR_OPTION`** indica que se produjo un error (o se cerró la ventana en el botón de la barra de título).
- **`EjemploJFileChooser1.java`**.

Cuadros de diálogo de `JFileChooser`

- Para obtener el nombre del fichero seleccionado se debe utilizar el método **`getSelectedFile`**.
- Si se permiten seleccionar múltiples ficheros, se puede utilizar **`getSelectedFiles`**, que devuelve un array de nombres de fichero.
- Los nombres de fichero se representan mediante objetos de la clase **`File`**.
- La clase `File` permite hacer diversas operaciones con el nombre de fichero que representa:
  - Saber si existe, si se puede escribir un archivo con ese nombre, si se puede leer, si es un directorio, etc.
  - Obtener el nombre del fichero, el *path*, los nombres de fichero de un directorio, etc.
  - Borrar, renombrar, cambiar los permisos, crear un fichero vacío.
- `JFileChooser` permite personalizar los cuadros de diálogo con otras opciones:

`http://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html`

Tablas de dos dimensiones: `JTable`

Tablas de dos dimensiones: `JTable`

- El componente **`JTable`** permite representar tablas bidimensionales de datos.
- Es un componente con gran cantidad de opciones.
- El usuario puede **editar** el contenido de las celdas de un `JTable`, **reordenar** las columnas arrastrándolas a la posición que desee, **ajustar el tamaño** de las columnas, etc.
- Normalmente el componente `JTable` se ajusta para ocupar todo el contenedor al que está asociado, y ajusta el tamaño de las celdas para representarlas todas en la ventana.
- Como en el caso de las listas, para mostrar las barras de desplazamiento vertical se debe utilizar un contenedor `JScrollPane`.
- En la barra de desplazamiento vertical, el comportamiento es el esperado.
- Pero aunque se utilice un `JScrollPane`, se debe fijar manualmente un ancho por defecto para las columnas y que no se redimensione la tabla para que tenga efecto (método setAutoResizeMode()).

Creación de un `JTable` vacío

- Se puede crear una tabla con las celdas vacías indicando en la constructora el número de filas y de columnas.
- En este caso se crea una tabla vacía **con celdas editables** y sin contenido.
- Se puede acceder a los datos que introduzca el usuario con **`Object getValueAt(int row, int col)`**.
- Se puede modificar el contenido de una celda con **`setValueAt(Object valor, int row, int col)`**.
- Por defecto, la selección de elementos es similar a un `JList`: por filas. Se puede modificar con los métodos correspondientes.
- **`EjemploJTable1.java`**.

Selección de elementos en un `JTable`

- Se puede decidir qué se va a mostrar como seleccionado en el `JTable` con los métodos:
  - **`setRowSelectionAllowed`** (por defecto) selecciona filas.
  - **`setColumnSelectionAllowed`** selecciona columnas.
  - **`setCellSelectionEnabled`** sólo va a permitir seleccionar celdas únicas.
- Para fijar la forma en que se seleccionan elementos en un `Jtable` se utiliza el método **`setSelectionMode`** con los siguientes valores:
  - **`ListSelectionModel.MULTIPLE_INTERVAL_SELECTION`**
  - **`ListSelectionModel.SINGLE_INTERVAL_SELECTION`**
  - **`ListSelectionModel.SINGLE_SELECTION`**
- Para obtener las filas/columnas seleccionadas (devuelven un array de enteros con los índices):
  - **`getSelectedRows`**
  - **`getSelectedColumns`**
- (`JTable` no está pensado para selección múltiple de celdas individuales).

Carga de datos en un `JTable`

- Los datos de la tabla se pueden asociar al componente visual de forma similar a como se hacía con las listas y combos.
- Se puede utilizar una constructora para crear una tabla a partir de un array de dos dimensiones para los datos y un array con los nombres de las columnas.
- Las celdas de la tabla son por defecto editables.
- Cuando se modifica un valor en la tabla, **se modifica también en el array asociado.**
- Todos los datos se tratan como *strings*, aunque se pueda especificar cualquier objeto.
- **`EjemploJTable3.java`**.
- También se puede especificar un **modelo de tabla.**

Uso de un modelo de tabla

- Como en el caso de listas y combos, se puede utilizar un **modelo de tabla.**
- Un modelo de tabla debe implementar el interfaz **`TableModel`**.
- También se puede utilizar un modelo de tabla por defecto: clase **`DefaultTableModel`**, ésta extiende a AbstractTableModel y ésta implementa la interfáz TableModel.
- DefaultTableModel contiene todos los métodos necesarios para modificar datos, incorporar más filas o más columnas.
- Podemos necesitar **crear un modelo de tabla específico**, mediante una clase que implemente el interfaz **`TableModel`**.

Uso de un modelo de tabla

- AbstractTableModel es una clase abstracta que proporciona una implementación por defecto de casi todos los métodos.
- Para crear un modelo de tabla no abstracto a partir de `AbstractTableModel` se deben implementar obligatoriamente los siguientes métodos:

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

- Los demás métodos se pueden **sobrescribir** para implementarlos como nos convenga.
- En el código que sobrescribe nuestros métodos, se realizan **operaciones sobre el modelo de datos.**
- Cuando se modifica algo del modelo, se debe **notificar a los observadores del modelo (el objeto `JTable`)** que ha habido cambios.
- **`Ejemplo`** EjJTablePR5.java

Uso de un modelo de tabla

- Los métodos que tiene `AbstractTableModel` para notificar los cambios de los datos a la tabla visual son los siguientes:
  - **`void fireTableCellUpdated(int row, int column)`**
    Notifica que ha cambiado la posición `row,column` de la tabla.
  - **`void fireTableDataChanged()`**
    Notifica que han cambiado los datos de la tabla, de forma que se refresque completamente la información del `JTable`.
  - **`void fireTableRowsDeleted(int firstRow, int lastRow)`**
    Notifica que han eliminado filas de la tabla.
  - **`void fireTableRowsInserted(int firstRow, int lastRow)`**
    Notifica que han insertado filas en la tabla.
  - **`void fireTableRowsUpdated(int firstRow, int lastRow)`**
  - **`void fireTableStructureChanged()`**
    Notifica que ha cambiado la estructura de la tabla: número de columnas, nombre de las columnas.
- Más información

`http://docs.oracle.com/javase/tutorial/uiswing/components/table.html`