11.E. Controles básicos.

Sitio: Aula Virtual CIERD (CIDEAD)
Curso: Programación_DAM
Libro: 11.E. Controles básicos.
Imprimido por: LUIS PUJOL
Día: miércoles, 20 de mayo de 2020, 09:20

Tabla de contenidos

1 Etiquetas y campos de texto.

Los **cuadros de texto** Swing vienen implementados en Java por la clase `JTextField`**.**

Para insertar un campo de texto, el procedimiento es tan fácil como: **seleccionar el botón correspondiente a** `JTextField` **en la paleta de componentes**, en el diseñador, y **pinchar sobre el área de diseño** encima del panel en el que queremos situar ese campo de texto. El tamaño y el lugar en el que se sitúe, dependerá del Layout elegido para ese panel.

El componente Swing etiqueta `JLabel`, se utiliza para crear etiquetas de modo que podamos insertarlas en un marco o un panel para visualizar un texto estático, que no puede editar el usuario. Los constructores son:

- `JLabel()`. Crea un objeto JLabel sin nombre y sin ninguna imagen asociada.
- `JLabel(Icon imagen)`. Crea un objeto sin nombre con una imagen asociada.
- `JLabel(Icon imagen, int alineacionHorizontal)`. Crea una etiqueta con la imagen especificada y la centra en horizontal.
- `JLabel(String texto)`. Crea una etiqueta con el texto especificado.
- `JLabel(String texto, Icon icono, int alineacionHorizontal)`. Crea una etiqueta con el texto y la imagen especificada y alineada horizontalmente.
- `JLabel(String texto, int alineacionHorizontal)`. Crea una etiqueta con el texto especificado y alineado horizontalmente.

Para saber más

En el siguiente enlace puedes ver cómo usar `DecimalFormat` para presentar un número en un `JTextField` o recoger el texto del `JTextField` y reconstruir el número.

Formatear cuadro de texto.

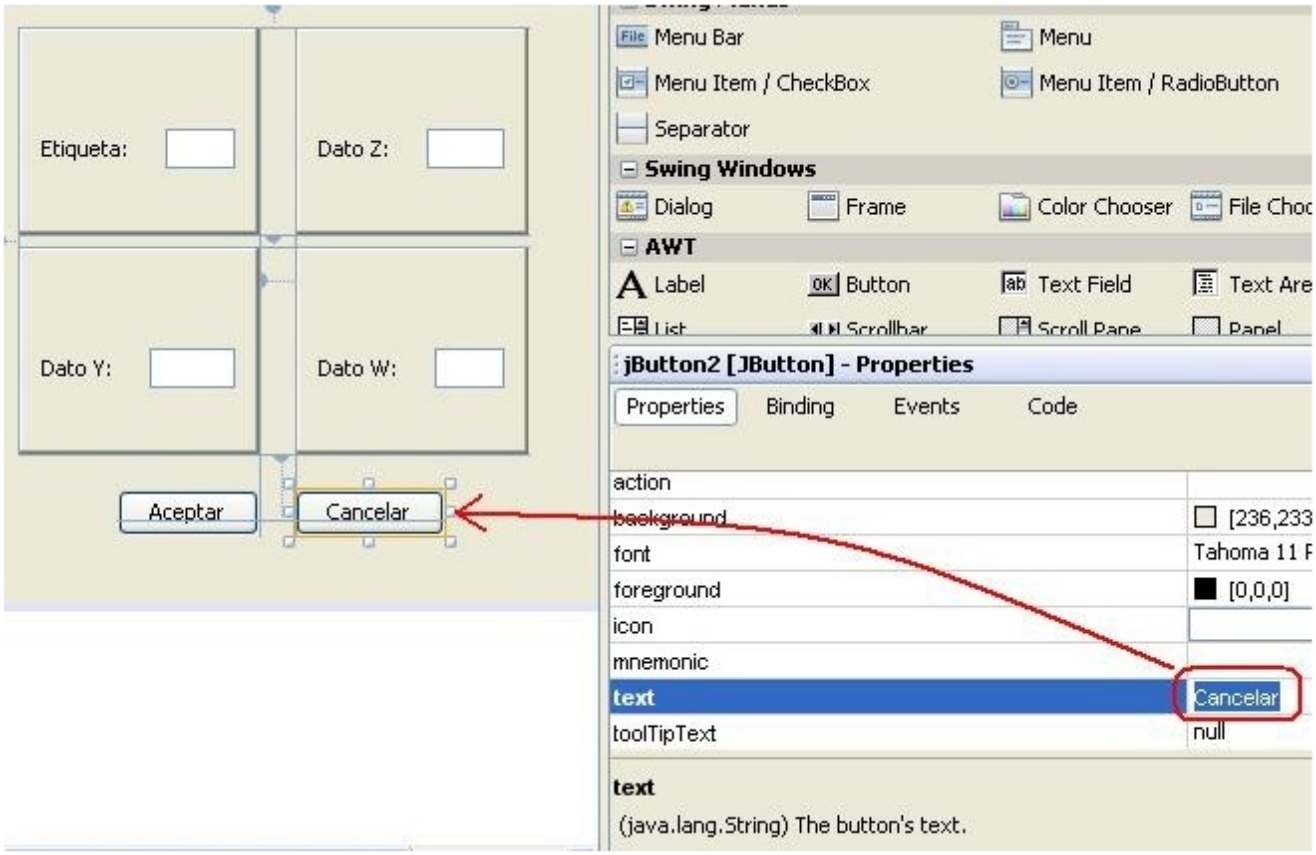
Autoevaluación

Un componente `JLabel` **permite al usuario de la aplicación en ejecución cambiar el texto que muestra dicho componente.**

- ○ Verdadero.
- ○ Falso.

2 Botones.

Ya has podido comprobar que prácticamente todas las aplicaciones incluyen botones que al ser pulsados efectúan alguna acción: hacer un cálculo, dar de alta un libro, aceptar modificaciones, etc.



Estos botones se denominan **botones de acción**, precisamente porque realizan una acción cuando se pulsan. En Swing, la clase que los implementa en Java es la `JButton`.

Los principales métodos son:

- `void setText(String`). Asigna el texto al botón.
- `String getText()` . Recoge el texto.

Hay un tipo especial de botones, que se comportan como **interruptores de dos posiciones** o estados (pulsados-on, no pulsados-off). Esos botones especiales se denominan botones on/off o `JToggleButton` .

Para saber más

A continuación puedes ver un enlace en el que se diseña una interfaz gráfica de usuario sencilla, con los controles que hemos visto hasta ahora.

Diseño de una GUI sencilla. (0.70 MB)

3 Casillas de verificación y botones de radio.

Las casillas de verificación en Swing están implementadas para Java por la clase `JCheckBox`, y los botones de radio o de opción por la clase `JRadioButton`. Los grupos de botones, por la clase `ButtonGroup`.

La funcionalidad de ambos componentes es en realidad la misma.

- Ambos tienen **dos "estados"**: seleccionados o no seleccionados (marcados o no marcados).
- Ambos **se marcan o desmarcan** usando el método `setSelected(boolean estado`**),** que establece el valor para su propiedad `selected`. (El estado toma el valor true para seleccionado y `false` para no seleccionado).
- A ambos le **podemos preguntar si están seleccionados o no**, mediante el método isSelected()**,** que devuelve `true` si el componente está seleccionado y `false` si no lo está.
- Para ambos **podemos asociar un icono distinto** para el estado de **seleccionado** y el de **no seleccionado**.

`JCheckBox` pueden usarse en menús mediante la clase `JCheckBoxMenuItem`.

`JButtonGroup` permite agrupar una serie de casillas de verificación (`JRadioButton`), de entre las que sólo puede seleccionarse una. Marcar una de las casillas implica que el resto sean desmarcadas automáticamente. La forma de hacerlo consiste en añadir un `JButtonGroup` y luego, agregarle los botones.

Cuando en **un contenedor** aparezcan **agrupados** varios **botones de radio** (o de opción), **entenderemos** que no son opciones independientes, sino que sólo uno de ellos podrá estar activo en cada momento, y necesariamente uno debe estar activo. Por tanto en ese contexto entendemos que son opciones excluyentes entre sí.

Para saber más

En el siguiente enlace puedes ver un vídeotutorial para crear un ejemplo básico de Java con interfaz gráfica.

Resumen textual alternativo

Autoevaluación

Los componentes radiobotones y las casillas de verificación tienen ambos dos estados: seleccionado y no seleccionado.

- Verdadero.
- Falso.

4 Listas (I).

En casi todos los programas, nos encontramos con que se pide al usuario que introduzca un dato, y además un dato de entre una lista de valores posibles, no un dato cualquiera.

La clase `JList` constituye un componente lista sobre el que se puede ver y seleccionar uno o varios elementos a la vez. En caso de haber más elementos de los que se pueden visualizar, es posible utilizar un componente `JScrollPanel` para que aparezcan barras de desplazamiento.

En los componentes `JList`, un modelo ListModel representa **los contenidos de la lista**. Esto significa que los datos de la lista se guardan en una estructura de datos independiente, denominada modelo de la lista. Es fácil mostrar en una lista los elementos de un vector o array de objetos, usando un constructor de `JList` que cree una instancia de `ListModel` automáticamente a partir de ese vector.

Vemos a continuación un ejemplo parar crear una lista `JList` que muestra las cadenas contenidas en el vector info[]:

```
String[] info = {"Pato", "Loro", "Perro", "Cuervo"};
```

```
JList listaDatos = new JList(info);
```

```
/* El valor de la propiedad model de JList es un objeto que proporciona una visión de sólo lectu
ra del vector info[].
```

```
El método getModel() permite recoger ese modelo en forma de Vector de objetos, y utilizar con lo
s métodos de la clase
```

```
Vector, como getElementAt(i), que proporciona el elemento de la posición i del Vector.  */
```

```
for (int i = 0; i < listaDatos.getModel().getSize(); i++) {
```

```
System.out.println(listaDatos.getModel().getElementAt(i));
```

```
}
```

Para saber más

A continuación puedes ver como crear un `JList`, paso a paso, en el enlace siguiente.

Crear un JList paso a paso.

5 Listas (II).

Cuando trabajamos con un componente `JList`, podemos seleccionar un único elemento, o varios elementos a la vez, que a su vez pueden estar contiguos o no contiguos. La posibilidad de hacerlo de una u otra manera la establecemos con la propiedad `selectionMode`.

Los valores posibles para la propiedad `selectionMode` para cada una de esas opciones son las siguientes constantes de clase del interface `ListSelectionModel`:

- `MULTIPLE_INTERVAL_SELECTION`: Es el valor por defecto. Permite seleccionar múltiples intervalos, manteniendo pulsada la tecla CTRL mientras se seleccionan con el ratón uno a uno, o la tecla de mayúsculas, mientras se pulsa el primer elemento y el último de un intervalo.

- `SINGLE_INTERVAL_SELECTION`: Permite seleccionar un único intervalo, manteniendo pulsada la tecla mayúsculas mientras se selecciona el primer y último elemento del intervalo.

- `SINGLE_SELECTION`: Permite seleccionar cada vez un único elemento de la lista.

Podemos establecer el valor de la propiedad `selectedIndex` con el método `setSelectedIndex()` es decir, el índice del elemento seleccionado, para seleccionar el elemento del índice que se le pasa como argumento.

Como hemos comentado, los datos se almacenan en un modelo que al fin y al cabo es un vector, por lo que tiene sentido hablar de índice seleccionado.

También se dispone de un método `getSelectedIndex()` con el que podemos averiguar el índice del elemento seleccionado.

El método `getSelectecValue()` devuelve el objeto seleccionado, de tipo `Object`, sobre el que tendremos que aplicar un "casting" explícito para obtener el elemento que realmente contiene la lista (por ejemplo un `String`). Observa que la potencia de usar como modelo un vector de `Object`, es que en el `JList` podemos mostrar realmente cualquier cosa, como por ejemplo, una imagen.

El método `setSelectedValue()` permite establecer cuál es el elemento que va a estar seleccionado.

Si se permite hacer selecciones múltiples, contamos con los métodos:

- `setSelectedIndices()`, al que se le pasa como argumento un vector de enteros que representa los índices a seleccionar.

- `getSelectedIndices()`, que devuelve un vector de enteros que representa los índices de los elementos o ítems que en ese momento están seleccionados en el `JList`.

- `getSelectedValues()`, que devuelve un vector de `Object` con los elementos seleccionados en ese momento en el `JList`.

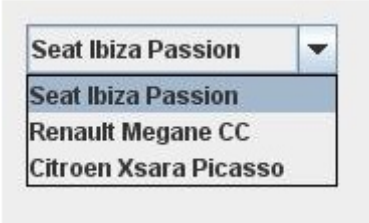
Autoevaluación

Los componentes `JList` permiten la selección de elementos de una lista, siempre que estén uno a continuación del otro de manera secuencial.

- Verdadero.
- Falso.

6 Listas desplegables.

Una **lista desplegable** se representa en Java con el componente Swing `JComboBox`. Consiste en una lista en la que sólo se puede elegir una opción. Se pueden crear `JComboBox` tanto editables como no editables.



Una lista desplegable es una mezcla entre un campo de texto editable y una lista. Si la propiedad `editable` de la lista desplegable la fijamos a verdadero, o sea a `true`, el usuario podrá seleccionar uno de los valores de la lista que se despliega al pulsar el botón de la flecha hacia abajo y dispondrá de la posibilidad de teclear directamente un valor en el campo de texto.

Establecemos la propiedad `editable` del `JComboBox` el método `setEditable()` y se comprueba con el método `isEditable()`. La clase `JComboBox` ofrece una serie de métodos que tienen nombres y funcionalidades similares a los de la clase `JList`.

Para saber más

En el siguiente enlace tienes algún ejemplo comentado para crear `JComboBox` por código.

Ejemplo `JComboBox`

7 Menús.

En las aplicaciones informáticas siempre se intenta que el usuario disponga de un menú para facilitar la localización una operación. La filosofía, al crear un menú, es que contenga todas las acciones que el usuario pueda realizar en la aplicación. Lo más normal y útil es hacerlo clasificando o agrupando las operaciones por categorías en submenús.

En Java usamos los componentes `JMenu` y `JMenuItem` para crear un menú e insertarlo en una barra de menús. La barra de menús es un componente `JMenuBar`. Los constructores son los siguientes:

- `JMenu()`. Construye un menú sin título.
- `JMenu(String s).` Construye un menú con título indicado por la cadena pasada como parámetro.
- `JMenuItem()`. Construye una opción sin icono y sin texto.
- `JMenuItem(Icon icono)`. Construye una opción con icono y con texto.

Podemos construir por código un menú sencillo como el de la imagen con las siguientes sentencias:

```
// Crear la barra de menú
```

```
JMenuBar barra = new JMenuBar();
```

```
// Crear el menú Archivo
```

```
JMenu menu = new JMenu("Archivo");
```

```
// Crear las opciones del menú
```

```
JMenuItem opcionAbrir = new JMenuItem("Abrir");
```

```
menu.add(opcionAbrir);
```

```
JMenuItem opcionguardar = new JMenuItem("Guardar");
```

```
menu.add(opcionguardar);
```

```
JMenuItem opcionSalir = new JMenuItem("Salir");
```

```
menu.add(opcionSalir);
```

```
// Añadir las opciones a la barra
```

```
barra.add(menu);
```

```
// Establecer la barra
```

```
setJMenuBar(barra);
```

Frecuentemente, dispondremos de alguna opción dentro de un menú, que al elegirla, nos dará paso a un conjunto más amplio de opciones posibles.

Cuando en un menú un elemento del mismo es a su vez un menú, se indica con una flechita al final de esa opción, de forma que se sepa que, al seleccionarla, nos abrirá un nuevo menú.

Para incluir un menú como submenú de otro basta con incluir como ítem del menú, un objeto que también sea un menú, es decir, incluir dentro del `JMenu` un elemento de tipo `JMenu`.

Autoevaluación

Un menú se crea utilizando el componente `JMenu` dentro de un `JList`.

- Verdadero.
- Falso.

7.1 Separadores.

A veces, en un menú pueden aparecer distintas opciones. Por ello, nos puede interesar destacar un determinado grupo de opciones o elementos del menú como relacionados entre sí por referirse a un mismo tema, o simplemente para separarlos de otros que no tienen ninguna relación con ellos.

El **componente Swing que tenemos en Java para esta funcionalidad es:** `JSeparator`, que dibuja una línea horizontal en el menú, y así separa visualmente en dos partes a los componentes de ese menú. En la imagen podemos ver cómo se han separado las opciones de Abrir y Guardar de la opción de Salir, mediante este componente.

Al código anterior, tan sólo habría que añadirle una línea, la que resaltamos en negrita:

```
...
```

```
menu.add(opcionguardar);
```

```
menu.add(new JSeparator());
```

```
JMenuItem opcionSalir = new JMenuItem("Salir");
```

```
...
```

Autoevaluación

En un menú en Java se debe introducir siempre un separador para que se pueda compilar el codigo.

- Verdadero.
- Falso.

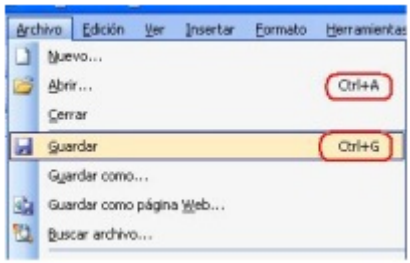
7.2 Aceleradores de teclado y mnemónicos.

A veces, tenemos que usar una aplicación con interfaz gráfica y no disponemos de ratón, porque se nos ha roto, o cualquier causa.

Además, cuando diseñamos una aplicación, debemos preocuparnos de las características de accesibilidad.

Algunas empresas de desarrollo de software obligan a todos sus empleados a trabajar sin ratón al menos un día al año, para obligarles a tomar conciencia de la importancia de que todas sus aplicaciones deben ser accesibles, usables sin disponer de ratón.

Ya no sólo en menús, sino en cualquier componente interactivo de una aplicación: campo de texto, botón de acción, lista desplegable, etc., es muy recomendable que pueda seleccionarse sin el ratón, haciendo uso exclusivamente del teclado.



Para conseguir que nuestros menús sean accesibles mediante el teclado, la idea es usar aceleradores de teclado o **atajos de teclado** y de **mnemónicos**.

Un acelerador o **atajo de teclado** es una combinación de teclas que se asocia a una opción del menú, de forma que pulsándola se consigue el mismo efecto que abriendo el menú y seleccionando esa opción.

Esa combinación de teclas **aparece escrita a la derecha de la opción del menú**, de forma que el usuario pueda tener conocimiento de su existencia.

Para añadir un atajo de teclado, lo que se hace es emplear la propiedad **accelerator** en el diseñador

Los atajos de teclado **no suelen** asignarse a todas las opciones del menú, sino **sólo a** las que más se usan.

Un **mnemónico** consiste en resaltar una tecla dentro de esa opción, mediante un subrayado, de forma que pulsando Alt + <el mnemónico> se abra el menú correspondiente. Por ejemplo en la imagen con Alt + A abriríamos ese menú que se ve, y ahora con Ctrl+G se guardaría el documento.

Para añadir mediante código un mnemónico a una opción del menú, se hace mediante la **propiedad mnemonic**. Los mnemónicos sí que deberían ser incluidos para todas las opciones del menú, de forma que todas puedan ser elegidas haciendo uso sólo del teclado, mejorando así la accesibilidad de nuestra aplicación. Para ver cómo se añadiría mediante el diseñador de NetBeans, mira el siguiente Debes conocer.

Debes conocer

En el siguiente enlace se puede ver cómo añadir a una aplicación que estemos construyendo con NetBeans, entre otras cosas, mnemónicos y aceleradores.

Diseñando con NetBeans

8 Ejemplo completo de creación de interfaces y control de eventos.

Para ver un ejemplo completo en Eclipse de creación de interfaces y control de eventos para una aplicación, a través del plugin WindowBuilder, puedes visualizar y hacer el ejemplo descrito en la siguiente colección de vídeos:

Video1:

Video2:

Video3:


Creación de interfaz gráfica en Java usando eclipse - parte 3