11.C. Generando programas en entorno gráfico.

Sitio: Aula Virtual CIERD (CIDEAD)
Curso: Programación_DAM
Libro: 11.C. Generando programas en entorno gráfico.
Imprimido por: LUIS PUJOL
Día: miércoles, 20 de mayo de 2020, 09:20

Tabla de contenidos

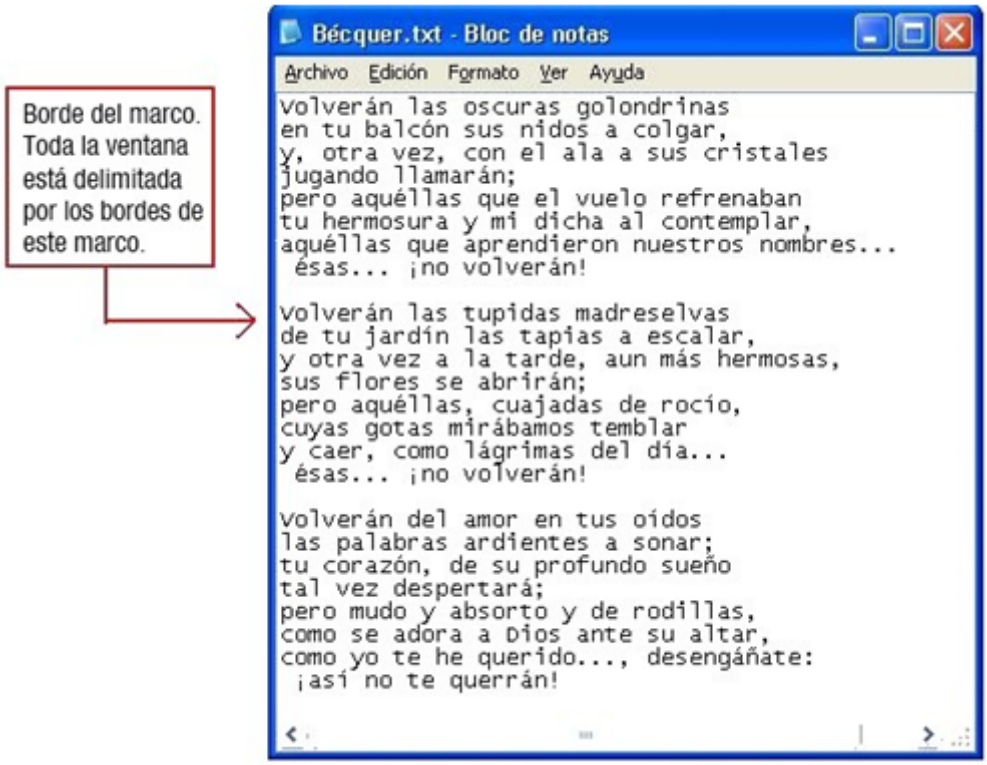
1 Generación de programas en entorno gráfico.

En este mismo tema, previamente, has visto la lista de los principales componentes Swing que se incluyen en la mayoría de las aplicaciones, junto con una breve descripción de su uso, y una imagen que nos dé una idea de cuál es su aspecto.

También hemos visto un ejemplo de cómo crear con ayuda de un IDE unos sencillos programas. Ahora, vamos a ver con mayor detalle, los componentes más típicos utilizados en los programas en Java y cómo incluirlos dentro de una aplicación.

1.1 Contenedores.

Por los ejemplos vistos hasta ahora, ya te habrás dado cuenta de la necesidad de que cada aplicación "contenga" de alguna forma esos componentes. ¿Qué componentes se usan para contener a los demás?



En Swing esa función la desempeñan un grupo de componentes llamados **contenedores** Swing.

Existen dos tipos de elementos contenedores:

- **Contenedores de alto nivel** o "peso pesado".
  - Marcos:JFrame **y** JDialog para aplicaciones
  - JApplet, para applets.

- **Contenedores de bajo nivel** o "peso ligero". Son los paneles: JRootPane y JPanel.

Cualquier aplicación, con interfaz gráfico de usuario típica, comienza con la apertura de una ventana principal, que suele contener la barra de título, los botones de minimizar, maximizar/restaurar y cerrar, y unos bordes que delimitan su tamaño.

Esa ventana constituye un marco dentro del cual se van colocando el resto de componentes que necesita el programador: menú, barras de herramientas, barra de estado, botones, casillas de verificación, cuadros de texto, etc.

Esa ventana principal o marco sería el contenedor de alto nivel de la aplicación.

Toda aplicación de interfaz gráfica de usuario Java tiene, al menos, un contenedor de alto nivel.

Los contenedores de alto nivel extienden directamente a una clase similar de AWT, es decir, `JFrame` extiende de `Frame`. Es decir, realmente necesitan crear una ventana del sistema operativo independiente para cada uno de ellos.

Los demás componentes de la aplicación no tienen su propia ventana del sistema operativo, sino que se dibujan en su objeto contenedor.

En los ejemplos anteriores del tema, hemos visto que podemos añadir un `JFrame` desde el diseñador de NetBeans o el WindowBuilder de Eclipse, o bien escribiéndolo directamente por código. De igual forma para los componentes que añadamos sobre él.

Para saber más

Te recomendamos que mires la siguiente web para ver información sobre `JFrame` y `JDialog`.

`JFrame` y `JDialog`

Autoevaluación

Cualquier componente de gráfico de una aplicación Java necesita tener su propia ventana del sistema operativo.

- Verdadero.
- Falso.

1.2 Cerrar la aplicación.

Cuando quieres terminar la ejecución de un programa, ¿qué sueles hacer? Pues normalmente pinchar en el icono de cierre de la ventana de la aplicación.

En Swing, una cosa es cerrar una ventana, y otra es que esa ventana deje de existir completamente, o cerrar la aplicación completamente.

- **Se puede** hacer **que una ventana no esté visible**, y sin embargo que ésta siga existiendo y ocupando memoria para todos sus componentes, usando **el método** `setVisible(false`). En este caso bastaría ejecutar para el JFrame el método `setVisible(true)` para volver a ver la ventana con todos sus elementos.

- **Si queremos cerrar la aplicación**, es decir, que no sólo se destruya la ventana en la que se mostraba, sino que se destruyan y liberen todos los recursos (memoria y CPU) que esa aplicación tenía reservados, tenemos que invocar al método `System.exit(0)` .

- **También se puede invocar para la ventana** `JFrame` **al método** `dispose()` , heredado de la clase `Window` , que no requiere ningún argumento, **y que borra todos los recursos de pantalla usados por esta ventana y por sus componentes, así como cualquier otra ventana que se haya abierto como hija de esta** (dependiente de esta). Cualquier memoria que ocupara esta ventana y sus componentes se libera y se devuelve al sistema operativo, y tanto la ventana como sus componentes se marcan como "no representables". Y sin embargo, el objeto ventana sigue existiendo, y podría ser reconstruido invocando al método `pack()` o la método `show()` , aunque deberían construir de nuevo toda la ventana.

Las ventanas `JFrame` de Swing permiten establecer una operación de cierre por defecto con el método `setDefaultCloseOperation()` , definido en la clase `JFrame` .
¿Cómo se le indica al método el modo de cerrar la ventana?

Los valores que se le pueden pasar como parámetros a este método son una serie de constantes de clase:

- `DO_NOTHING_ON_CLOSE` : **no hace nada**, necesita que el programa maneje la operación en el método `windowClosing()` de un objeto `WindowListener` registrado para la ventana.

- `HIDE_ON_CLOSE` : **Oculta** de ser mostrado en la pantalla pero no destruye el marco o ventana después de invocar cualquier objeto `WindowListener` registrado.

- `DISPOSE_ON_CLOSE` : **Oculta y termina (destruye) automáticamente el marco o ventana** después de invocar cualquier objeto `WindowListener` registrado.

- `EXIT_ON_CLOSE:` Sale de la aplicación usando el método `System.exit(0)`. Al estar definida en `JFrame` , se puede usar con aplicaciones, pero no con applets.

Autoevaluación

`System.exit(0)` **oculta la ventana pero no libera los recursos de CPU y memoria que la aplicación tiene reservados.**

- Verdadero.
- Falso.