MEDAC
Instituto Oficial de Formación Profesional

PROGRAMACIÓN

Interfaces gráficas de usuario II

16

ÍNDICE

OBJETIVOS

Elaborar programas con interfaz gráfica.

Conocer los diferentes elementos de la interfaz gráfica.

Trabaja con eventos.

Gestiona interfaces gráficas.

Realiza programas con más de una ventana gráfica.

/ 1. Introducción y contextualización práctica

En esta unidad vamos a ampliar nuestros conocimientos sobre interfaces gráficas.

Vamos a descubrir que muchos elementos que vamos a necesitar en nuestros desarrollos, ya están implementados y no tendremos que diseñarlos de cero, como son, por ejemplo, las ventanas de diálogo y las ventanas de selección de ficheros (las típicas que aparecen al pulsar 'abrir' en cualquier editor de texto).

También vamos a ver cómo podremos implementar menús en nuestras interfaces.

Por último, aprenderemos cómo crear varias ventanas en una misma aplicación, y hacer que se conecten entre ellas.

Escucha el siguiente audio en el que planteamos el caso práctico que iremos resolviendo a lo largo de esta unidad.

Fig. 1. Diseñando interfaces

Audio Intro. "Uso de la interfaz gráfica avanzado"

http://bit.ly/3ogic8z

/ 2. Ventanas de diálogo predefinidas: Ventana de información

Hasta el momento todos los mensajes de error, notificaciones y demás, los hemos estado mostrando por consola, aspecto que para el desarrollador del programa no presenta inconveniente, pero para el usuario de la aplicación no es adecuado.

Este problema lo podemos solucionar de una forma muy fácil con las **ventanas de diálogo predefinidas.**

Este tipo de ventanas consisten en diálogos que ya están **predefinidos**, implementados y listos para mostrarse en cualquier punto de nuestro programa, y que **mostrarán** un mensaje de error, una notificación o cualquier otro tipo de **aviso**, sea una tarea muy sencilla de implementar.

La primera ventana de diálogo predefinida que vamos a estudiar es la **ventana de información**. Mediante ésta, podremos notificar un mensaje al usuario.

Para crear esta ventana escribiremos el siguiente código:

```
String mensaje = "Mensaje de información";

JOptionPane.showMessageDialog(this, mensaje);
```

Código 1. Creando un mensaje de información.

Al haber utilizado la clase *JOptionPane*, la tendremos que importar.

El funcionamiento es muy simple, básicamente, utilizamos el método ***showMessage*** al que le pasamos dos parámetros:

- ***this:*** Es la ventana padre de la que dependerá la ventana de información. En este parámetro siempre pondremos this.
- ***mensaje:*** Es el mensaje que queremos mostrar.

El resultado de ejecutar el código es el siguiente:

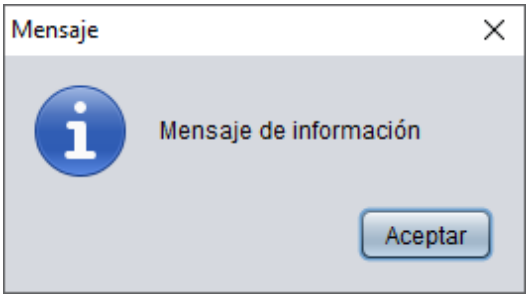

Fig. 2. Diálogo de información.

Este tipo de **ventanas** se conocen como **modales**, esto quiere decir que **hasta que no cerremos el mensaje no podremos interactuar con nuestro programa.**

/ 3. Ventanas de diálogo predefinidas: Ventana de peligro

Algo muy útil a la hora de utilizar las ventanas de diálogo predefinidas es la posibilidad de mostrar mensajes de peligro, es decir, **advertencias** para el usuario.

Mediante la ventana de peligro podremos hacer esto de una forma muy sencilla.

Para crear esta ventana escribiremos el siguiente código:

```
String mensaje = “Mensaje de peligro”;
String titulo = “Peligro”;
JOptionPane.showMessageDialog(this,
        mensaje,
        titulo,
        JOptionPane.WARNING_MESSAGE);
```

Código 2. Creando un mensaje de peligro.

El funcionamiento se basa en utilizar el método ***showMessage*** al que le pasamos cuatro parámetros:

- ***this:*** Es la ventana padre de la que dependerá la ventana de información. En este parámetro siempre pondremos this.
- ***mensaje*:** Es el mensaje que queremos mostrar.
- ***título*:** Es el título que queremos que tenga la ventana que vamos a mostrar.
- ***JOptionPane.WARNING_MESSAGE:*** Con esto indicamos que es una ventana del tipo peligro y mostrará el icono que le corresponde.

El resultado de ejecutar ese código es el siguiente:

Fig. 3. Diálogo de peligro.

Audio 1. “Pequeñas variaciones en los iconos”

http://bit.ly/3nbIZ4p

/ 4. Caso práctico 1: "Mensajes con interfaz gráfica"

Planteamiento: Pilar y José están realizando un nuevo ejercicio que han de entregar este mismo fin de semana.

Este ejercicio consiste en mostrar una pantalla con una serie de opciones, tipo encuesta, las cuales podrá seleccionar el usuario según sus gustos.

Las preguntas que se mostrarán serán ¿cuál es tu sistema operativo favorito?, pudiendo elegir entre Linux, Windows y Mac, ¿cuál es tu especialidad?, pudiendo elegir entre programación, diseño gráfico y administración, y una pregunta de cuántas horas semanales le dedica a ello.

Como con interfaz gráfica no podemos utilizar la línea de salida para mostrar texto, el resultado se deberá mostrar mediante un mensaje.

Nudo: ¿Cómo podrían nuestros amigos implementar esto? ¿Deberán lanzar varios mensajes unos a continuación de otros para mostrar toda la información recogida?

Desenlace: Cuando utilizamos ventanas de mensajes para mostrar información lo idóneo es mostrar una única ventana con un único mensaje.

En el caso que tienen que resolver nuestros amigos hay tres datos que deberán mostrar: el sistema operativo preferido, la especialidad y las horas dedicadas.

Para ello, deberemos crear un único *String* que contenga toda la información. En este *String* podemos utilizar el separador \n para que la información se muestre en líneas diferentes, visualizándose así la información de una forma mucho más ordenada y fácil de leer.

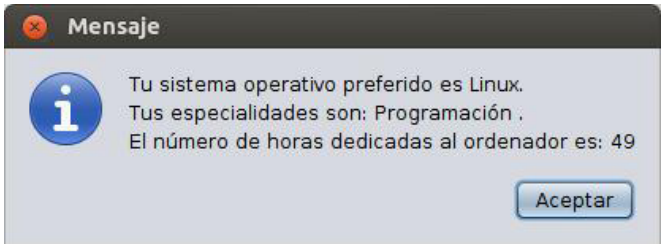

Fig. 4. Ejemplo de mensaje con varias líneas de texto.

/ 5. Ventanas de diálogo predefinidas: Ventana de error

La siguiente ventana predefinida que vamos a ver es la ventana de error.

Mediante esta ventana podremos **informar al usuario de los errores ocurridos en el programa.** Este tipo de ventana la podremos utilizar, por ejemplo, dentro de los **bloques *catch*** para informar de las excepciones.

Para crear esta ventana escribiremos el siguiente código:

```
String mensaje = "Mensaje de error";
String titulo = "Error";
JOptionPane.showMessageDialog(this,
        mensaje,
        titulo,
        JOptionPane.ERROR_MESSAGE);
```

Código 3. Creando un mensaje de error

Su funcionamiento se basa en el uso del método ***showMessage*** al que le pasamos cuatro parámetros:

- ***this:*** Es la ventana padre de la que dependerá la ventana de información. En este parámetro siempre pondremos this.
- ***mensaje:*** Es el mensaje que queremos mostrar.
- ***título:*** Es el título que queremos que tenga la ventana que vamos a mostrar.
- ***JOptionPane.ERROR_MESSAGE:*** Con esto indicamos que es una ventana de tipo error y mostrará el icono que le corresponde.

El resultado de ejecutar ese código es el siguiente:

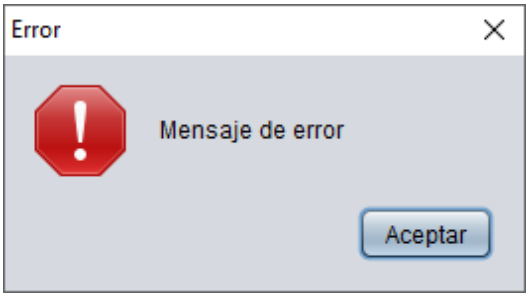

Fig. 5. Diálogo de error.

/ 6. Ventanas de diálogo predefinidas: Ventana personalizada

Ya hemos comprobado que existen ventanas de diálogo predefinidas, pero **¿podemos crear nuestras propias ventanas de diálogo de forma personalizada**?

La respuesta es que sí. **Para ello podremos utilizar el siguiente código:**

```
// Creamos el icono para mostrar
String r_icono = “imagenes/emoticono_sonrisa.png”;
ImageIcon icono = new
ImageIcon(getClass().getResource(r_icono));
// Título personalizado, icono personalizado
JOptionPane.showMessageDialog(this,
        “Mensaje de texto normal.”,
        “Título”,
        JOptionPane.INFORMATION_MESSAGE,
        icono);
```

Código 4. Creando un mensaje de información personalizado

El funcionamiento es muy sencillo:

Primero **creamos un objeto** de la clase ***ImageIcon*** a partir de la ruta del icono que queremos mostrar.

Posteriormente, utilizamos el método ***showMessage*** al que le pasamos cinco parámetros:

- ***this:*** Es la ventana padre de la que dependerá la ventana de información. En este parámetro siempre pondremos this.
- ***mensaje***: Es el mensaje que queremos mostrar.
- ***título:*** Es el título que queremos que tenga la ventana que vamos a mostrar.
- ***JOptionPane.INFORMATION_MESSAGE:*** Con esto indicamos que es una ventana del tipo información.
- ***icono:*** Será el icono que queremos que aparezca.

El resultado de ejecutar ese código es el siguiente:

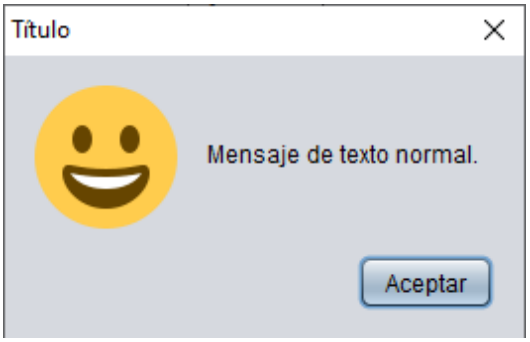

Fig. 6. Diálogo personalizado.

/ 7. Ventanas de diálogo predefinidas: Ventana de pregunta

La personalización de las ventanas puede ir más allá de simplemente cambiar el icono.

Podemos crear una ventana en la que aparezca una pregunta y el usuario pueda elegir una respuesta, obtenerla esta información, y actuar en consecuencia. **Para ello podremos utilizar el siguiente código:**

```
// Texto de las opciones para responder
Object[] opciones = {“Si”, “No”};
int respuesta = JOptionPane.showOptionDialog(this,
        “¿Desea guardar los cambios realizados?”,
        “Pregunta”,
        JOptionPane.YES_NO_CANCEL_OPTION,
        JOptionPane.QUESTION_MESSAGE,
        null,
        opciones,
        opciones[0]);
System.out.println(“Has respondido “ + respuesta);
```

Código 5. Creando un mensaje de pregunta personalizado.

Para implementarla, primero creamos un ***array* con las posibles respuestas a mostrar** y luego utilizamos el método ***showOptionDialog*** al que le pasamos los siguientes parámetros:

- ***this:*** Es la ventana padre de la que dependerá la ventana de información. En este parámetro siempre pondremos this.
- ***mensaje:*** Es el mensaje que queremos mostrar.
- ***título:*** Es el título que queremos que tenga la ventana que vamos a mostrar.
- ***JOptionPane.YES_NO_CANCEL_OPTION:*** Con esto indicamos que es una ventana del tipo pregunta.
- ***JOptionPane.QUESTION_MESSAGE:*** Mediante esta sentencia hacemos que muestre el icono de una pregunta.
- ***opciones:*** Serán las opciones a mostrar.
- ***opciones[0]:*** Será la opción que aparecerá señalada

Con ***System.out.println*** situado al final podremos ver qué respuesta ha elegido el usuario.

El resultado de ejecutar ese código es el siguiente:

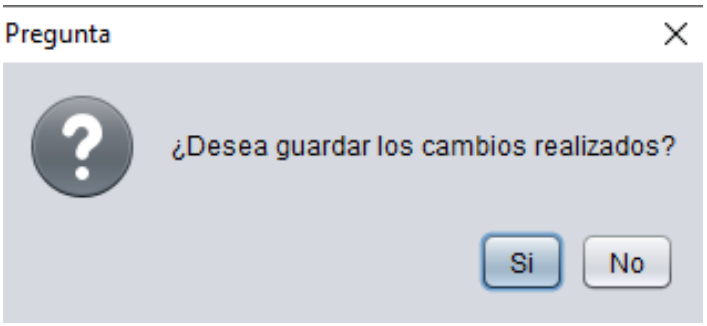

Fig. 7. Diálogo personalizado de pregunta.

/ 8. Selección de ficheros: Ventana de abrir fichero

Seguramente habrás utilizado multitud de programas que permiten realizar la opción de abrir un fichero.

Para realizar esto, se muestra **una ventana que nos permitirá navegar por el sistema de ficheros de nuestro ordenador, pudiendo seleccionar el fichero que necesitemos abrir.**

Este tipo de ventanas son otro ejemplo de ventana que ya está implementada y lista para que la invoquemos cuando sea necesario.

Para ello necesitamos podemos **usar el siguiente código**, en el que podrás encontrar a través de los comentarios, qué realiza cada línea:

```
// Creamos un JFileChooser, es una ventana para seleccionar un fichero
JFileChooser fileChooser = new JFileChooser();
// Hacemos que aparezca un fichero por defecto
fileChooser.setSelectedFile(new File("fichero.txt"));
// Cambiamos el título de la ventana
fileChooser.setDialogTitle("Selecciona un fichero");
// Agregamos dos filtros al selecctor de ficheros
FileFilter filter = new FileNameExtensionFilter("Ficheros de texto", "txt");
fileChooser.addChoosableFileFilter(filter);
FileFilter filter2 = new FileNameExtensionFilter("Ficheros java", "java");
fileChooser.addChoosableFileFilter(filter2);
// Mostramos la ventana de seleccionar el fichero
int resultado = fileChooser.showOpenDialog(null);
// Si se ha seleccionado un fichero, si se pulsa cancelar no se ejecuta
if (resultado == JFileChooser.APPROVE_OPTION)
{
// Obtenemos la ruta del fichero seleccionado
String ruta = fileChooser.getSelectedFile().getAbsolutePath();
}
```

Código 6. Código para crear una ventana de selección de fichero.

Vídeo 1. "Seleccionando ficheros"
https://bit.ly/395YqpI

/ 9. Selección de ficheros: Ventana de guardar fichero

Además de poder crear una ventana de selección de fichero, otro tipo que podemos crear e**s la ventana que aparece en las opciones de 'guardar'**, tal y como estamos acostumbrados a utilizar en casi cualquier programa cuando nos disponemos a guardar los trabajos que estemos desarrollando en dicha aplicación.

Esta ventana nos permitirá seleccionar un fichero para guardar un contenido que tengamos en nuestro programa. **Para ello escribiremos el siguiente código:**

```
// Creamos un JFileChooser, es una ventana para seleccionar un fichero
JFileChooser fileChooser = new JFileChooser();
// Hacemos que aparezca un fichero por defecto
fileChooser.setSelectedFile(new File("fichero.txt"));
// Cambiamos el título de la ventana
fileChooser.setDialogTitle("Selecciona un fichero");
// Mostramos la ventana de seleccionar el fichero
int resultado = fileChooser.showSaveDialog(null);
// Si se ha seleccionado un fichero, si se pulsa cancelar no se ejecuta
if (resultado == JFileChooser.APPROVE_OPTION)
{
// Obtenemos la ruta del fichero
String ruta = fileChooser.getSelectedFile().getAbsolutePath();
}
```

Código 7. Código para crear una ventana de guardar fichero.

Podemos ver que el código es muy parecido al de la ventana de abrir. A este tipo de ventanas también podremos agregarle tantos filtros de ficheros como deseemos. El resultado de la ejecución de este código es el siguiente:

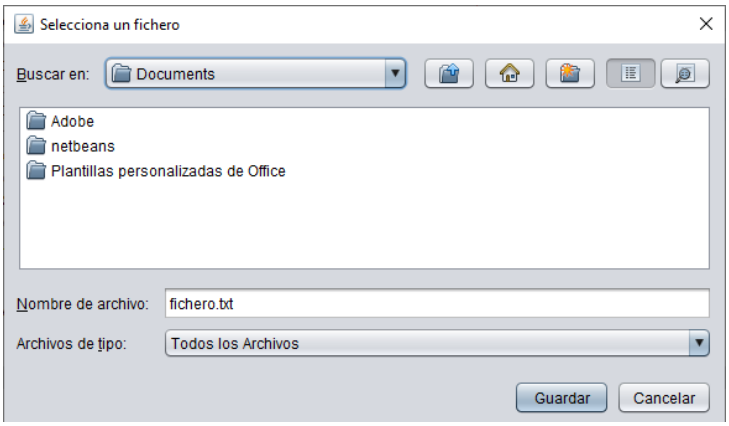

Fig. 8. Ventana para guardar un fichero.

/ 10. Menús

Mediante el uso de los menús vamos a pode**r agregar a nuestros programas el menú convencional que aparecen en todas las aplicaciones**. Podremos también, **crear submenús** dentro de los menús y darles una funcionalidad, como si de un botón de tratase.

Los menús van a permitir que podamos agrupar funcionalidades y organizarlas de una forma sencilla y rápida de encontrar para el usuario.

Los menús se encuentran **en la paleta de elementos de la biblioteca *SWING*** en la categoría "***Swing Menus***". Esta categoría suele venir compactada, pero basta con pulsar el símbolo más (+) y se mostrarán todos los elementos de los que se dispone.

Para crear menús en Java tendremos las siguientes clases:

- ***JMenuBar:*** Esta clase representa el menú completo de la aplicación.
- ***JMenu:*** Clase que representa un menú concreto.
- ***JMenuItem:*** Ésta representa un ítem de un menú, al que le podremos dar funcionalidad.
- ***JMenuBoxMenuItem:*** Igualmente, representa un ítem de un menú, pero mostrará un check box en él.
- ***JRadioButtonMenuItem:*** También representa un ítem de un menú, pero mostrará un radio button en él.
- ***Separator:*** Esta clase representa un separador entre menús.

Dentro de un menú podremos tener elementos de menú convencionales como *CheckBox, RadioButton* y separadores.

Podremos disponer de un menú con un submenú, simplemente añadiendo un menú dentro de otro.

Podremos anidar tantos menús dentro de otros como queramos, no obstante, a efectos de usabilidad por parte de los usuarios, hay que tener cuidado con esto, **y no formar árboles de menú de varios niveles**, ya que la organización de la información de la aplicación, se volvería complicada de entender y de consultar.

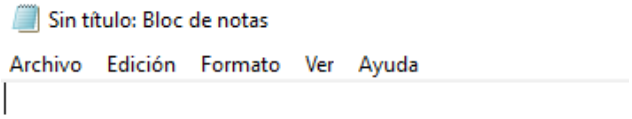

Fig. 9. Ejemplo de menú.

/ 11. Menús: Elementos de menús

Ya hemos visto diferentes tipos de menús que podemos poner en nuestros programas.

Para agregar un menú a nuestro menú de la aplicación que estemos desarrollando, seleccionamos el elemento ***Menu*** y lo arrastraremos hasta el menú de la aplicación. Podremos ver que **en árbol de elementos del programa aparecerá el menú que acabamos de agregar.** A éste le podremos cambiar su texto a mostrar.

Para agregar un *ítem* a un menú seleccionaremos el tipo de ítem que queremos agregar: *Menu Item, Menu Item CheckBox o Menu Item Radio Butto*n, y lo arrastraremos dentro del menú donde necesitemos que aparezca. Una vez agregado, aparecerá dentro del árbol de componentes del programa y también podremos cambiarle el texto a mostrar.

Para agregar un separador entre los elementos de un menú, elegimos *Separador* en la paleta de menús y lo arrastramos hacia el punto donde deseemos que se coloque.

A los menús podremos **darle una funcionalidad** que ejecutarán cuando se pulse sobre ellos, para esto haremos doble click sobre el menú al que queramos darle funcionalidad, y se le agregará el **evento *actionPerformed,*** en el que escribiremos el código que se ejecutará al pulsar sobre el menú.

A los menús, igual que al resto de elementos, deberemos darles un nombre para poder distinguirlos. En este caso podremos aplicar la siguiente regla:

- ***JMenuItem:*** Su nombre empezará por 'm' de menú y la función que realice.

A los menús podremos asignarles también un **"*shortcut*" o acceso directo mediante una combinación de teclas** que hará que, al pulsarlas, se ejecute la acción del menú. Para ello seleccionamos el menú deseado y aparecerá la opción "*shortcut*", donde al hacer doble *click*, nos aparecerá un cuadro para configurar el acceso directo, pudiendo utilizar las teclas *Ctrl, Alt, y Shift* junto a una tecla más.

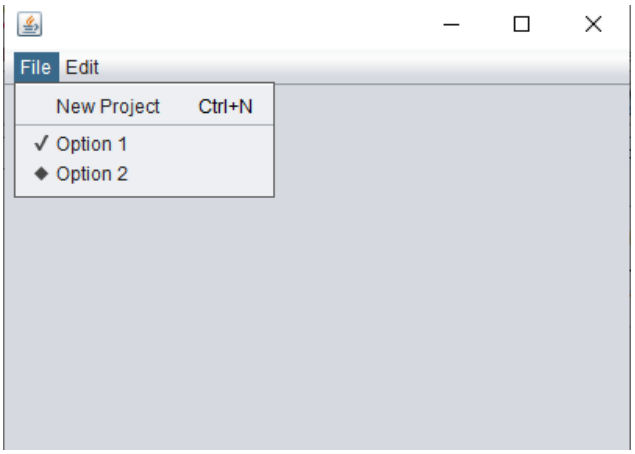

Fig. 10. Diferentes menús.

/ 12. Caso práctico 2: "Abuso de los menús"

Planteamiento: Nuestros amigos, en el estudio de la asignatura de Programación, se encuentran repasando el concepto e implementación de menús y sus distintas funcionalidades.

"Esto es genial, no pensé que fuera tan sencillo crear menús y poder darles funcionalidad, los voy a utilizar para todo", dice Jose muy contento.

Pilar, en cambio, cree que los menús son de utilidad, pero haciendo un poco de memoria, no cree haberlos visto implementados en algunas de las aplicaciones que ha estado usando. "Yo creo que tendríamos que replantearnos el uso de los menús, pienso que hay que usarlos con moderación", le dice Pilar a José.

Nudo: ¿Qué piensas al respecto? ¿Crees que tiene razón Pilar y hay que usar los menús para cualquier funcionalidad que tengamos en nuestra aplicación? ¿O en cambio, crees que José tiene razón y hay que plantearse su uso?

Desenlace: Es cierto que los menús son algo muy sencillo de utilizar y que nos ayudan en gran medida a organizar la información, además, nos pueden sacar de más de un apuro cuando no veamos con claridad dónde poner ubicar una opción.

En los programas y aplicaciones actuales, es cierto que se siguen utilizando, pero poco a poco están desapareciendo en ciertos entornos, como en los programas que utilizan una pantalla táctil, ya que ahí, un menú puede ser algo poco útil. En estos casos es mejor usar botones y estudiar detenidamente qué funcionalidades deberemos poner.

Así que, en cierto modo, tanto la reflexión de Pilar como la de José es acertada.

Se pueden usar menús en programas de escritorio, pero no sería muy recomendable usarlos en programas que se vayan a usar en interfaces táctiles, por ejemplo.

Fig. 11. Menús sí o no...

/ 13. Creación de varias ventanas

Cualquier programa o aplicación en la actualidad, por sencillo que sea, **constará de varias ventanas y podrá realizarse una navegación entre las mismas**, haciendo que se llamen entre sí.

Para crear una ventana nueva en nuestro programa deberemos crear un ***JFrame* nuevo**.

Una vez creado veremos que tenemos la configuración de elementos con la que ya estamos familiarizados, pudiendo crear una interfaz con todos los elementos ya estudiados y sus *layouts*.

Un dato importante en la nueva ventana es que tendremos que **modificar su operación** por defecto al cerrarse.

Si no la modificamos, su valor por defecto será cerrar todo el programa, así que tendremos que hacer que cuando pulsemos el botón de cerrar de la ventana, únicamente se cierre ella.

Para ello, en la ventana de propiedades de la nueva ventana, seleccionamos en la opción ***defaultCloseOperation*** el valor DISPOSE. Esto hará que cuando cerremos dicha ventana se cierre ella únicamente y no el programa completo.

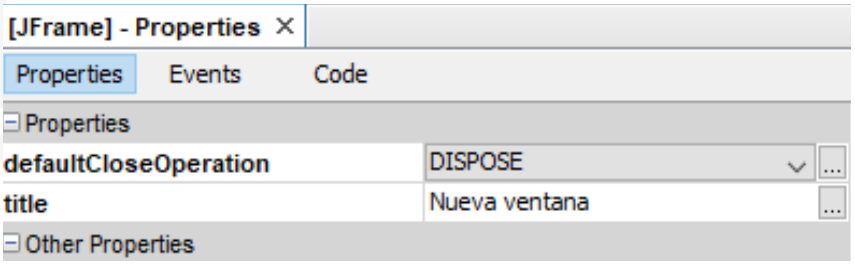

Fig. 12. Opción DISPOSE.

Una vez creada la ventana nueva, al ejecutar el programa observaremos que la que se ejecuta es la primera que se creó, ya que es la ventana principal, pero, ¿cómo podremos hacer para que aparezca nuestra nueva ventana?

Para poder lanzar la nueva ventana tendremos que crearle un objeto y llamar a los siguientes métodos:

```
// Creamos un objeto de la ventana

NuevaVentana ventana = new NuevaVentana();

// Hacemos que se muestre la ventana

ventana.setSize(200, 200);

ventana.setVisible(true);
```

Código 8. Código para mostrar una ventana nueva.

Mediante el método ***setVisible*** haremos que la ventana que hemos creado se muestre., y con el método ***setSize*** podremos modificar el tamaño de la ventana.

/ 14. Comunicación de varias ventanas

Cuando trabajamos con varias pantallas en un programa puede que sea necesario hacer que haya comunicación entre ellas, es decir, que de una pantalla (ventana) se pase información a la siguiente.

Un ejemplo muy claro de esto es cuando estamos registrándonos en un programa y hay varias pantallas para dicho registro. En esta situación, podemos observar que al introducir un dato y pulsar 'siguiente', dependiendo de ese dato, aparecen unos resultados u otros en la nueva pantalla.

Esto es posible ya que la primera pantalla donde introducimos el dato, le pasó la información a la siguiente pantalla, pudiendo ésta mostrar la información correspondiente al dato introducido anteriormente.

Para poder implementar estas características necesitaremos seguir **los siguientes pasos:**

- En la ventana nueva, necesitaremos **crear una variable de clase** con el tipo de dato que le vamos a pasar.
- También en la ventana nueva, crearemos **un nuevo constructor con parámetros**, al que le pasaremos el dato que necesitemos. En este constructor deberemos llamar al método ***initComponentes*** en primer lugar, para que la ventana gráfica se cree correctamente. Una vez hecho esto ya escribiremos el código correspondiente a la acción que queramos implementar.
- En la ventana principal, la que va a llamar a la nueva, cuando creemos el objeto de la nueva ventana **tendremos que utilizar el constructor con parámetros nuevo que acabamos de crear, pasándole el valor que necesitemos.** Después llamaremos al método ***setVisible*** de forma habitual.

Una vez hecho esto, cuando llamemos a la nueva ventana, le estaremos pasando la información pertinente y se mostrará un resultado dependiendo de ello.

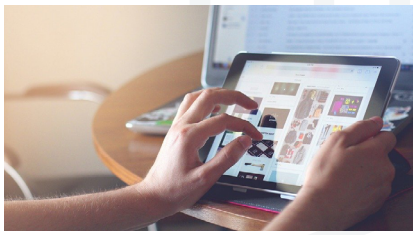

Fig. 13. Comunicación entre ventanas.

Vídeo 2. "Comunicación entre ventanas"
https://bit.ly/3ohAPsO

/ 15. Resumen y resolución del caso práctico de la unidad

A lo largo de esta unidad hemos podido comprobar que existen varios tipos de **ventanas ya diseñadas** y listas para usarse: las ventanas de diálogo predefinidas, que sirven para mostrar mensajes al usuario.

Hemos estudiado también que este tipo de ventanas podemos **personalizarlas** e incluso hacer que sirvan para realizar preguntas y obtener las respuestas.

También hemos aprendido cómo podemos **integrar menús** con sus respectivos submenús en nuestros programas de una forma muy sencilla y rápida, pudiendo hacer que dispongan de funcionalidad al seleccionarlos.

Por último, hemos visto cómo podemos crear **más de una ventana** en nuestro programa, cómo llamarlas, y hacer que se intercambien información entre ellas.

Resolución del caso práctico inicial

Cuando empezamos con el diseño de interfaces, lo habitual es tener una única ventana en las aplicaciones, ya que se están dando los primeros pasos y los programas no deberían ser muy complicados.

Al ir avanzando en el estudio, ya notamos que con una única ventana estaríamos muy limitados en el desarrollo, ya que casi cualquier programa con miras a cubrir una necesidad real, por simple y sencillo que sea, va a necesitar de varias ventanas, que, incluso, se comuniquen entre ellas.

Para poder crear más de una ventana, tal y como hemos visto en la unidad, basta con crear un nuevo *JFrame,* al igual que hemos hecho al crear el proyecto, creamos un objeto de la ventana y hacemos que se muestre la ventana. Una vez hecho esto tendremos disponible una nueva ventana en nuestro proyecto.

Fig 14. Creación de varias ventanas.

/ 16. Bibliografía

How to Make Dialogs (The JavaTM Tutorials > Creating a GUI With JFC/Swing > Using Swing Components). (s. f.). docs.oracle.com. Recuperado 16 de septiembre de 2020, de https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html

How to Use Menus (The JavaTM Tutorials > Creating a GUI With JFC/Swing > Using Swing Components). (s. f.). docs.oracle.com. Recuperado 16 de septiembre de 2020, de https://docs.oracle.com/javase/tutorial/uiswing/components/menu.html