MEDAC
Instituto Oficial de Formación Profesional

PROGRAMACIÓN

Interfaces gráficas de usuario I

15

ÍNDICE

OBJETIVOS

Elaborar programas con interfaz gráfica.

Conocer los diferentes elementos de la interfaz gráfica.

Trabajar con eventos.

Gestionar interfaces gráficas.

/ 1. Introducción y contextualización práctica

En esta unidad vamos a aprender cómo podremos crear un proyecto en *NetBeans* con interfaz gráfica.

También vamos a ver los elementos más básicos con los que contaremos, y que nos servirán para realizar la gran mayoría de las interfaces que necesitemos.

Estudiaremos cómo podemos utilizarlos y cómo se les podrá otorgar una funcionalidad concreta, por ejemplo, que al pulsar un botón se produzca un evento de algún tipo.

Por último, aprenderemos las técnicas para que los elementos de nuestras interfaces se adapten a las mismas.

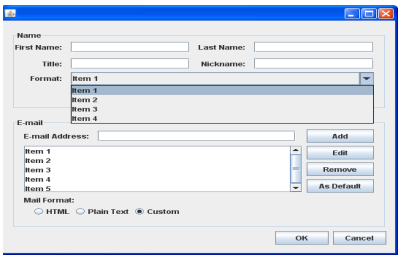

Fig. 1. Las interfaces gráficas de usuario (GUI) permiten al usuario comunicarse con la máquina y a esta presentar los resultados del procesamiento de la información.

Por ahora, escucha el siguiente audio en el que planteamos el caso práctico que iremos resolviendo a lo largo de esta unidad.

Audio Intro. "El uso de la interfaz gráfica"
http://bit.ly/391AvYK

/ 2. Creación de un proyecto con interfaz gráfica

Hasta el momento, nuestros **programas siempre han funcionado** de la misma forma, pidiendo **información** al usuario para que la introduzca **por teclado**, procesando dicha información, y mostrando por **pantalla el resultado** del procesamiento.

Ahora vamos a ver cómo podemos hacer que nuestros **programas tengan una interfaz gráfica,** donde podremos colocar elementos para que el usuario interactúe con ellos, tales como botones, cajas de texto, etiquetas, tablas, etc. A partir de este momento, nuestros programas dejarán de usar la consola para pasar a tener una interfaz gráfica totalmente funcional.

Para la creación de las interfaces gráficas en Java vamos a utilizar la **biblioteca *SWING***, la cual nos va a proveer de todo lo necesario para poder crear ventanas totalmente funcionales de una forma rápida y sencilla. Esta biblioteca ya viene integrada en la propia **JDK de Java**, así que **no tendremos que instalar nada para poder utilizarla.**

Para crear un proyecto con interfaz gráfica seguiremos los siguientes pasos:

1. Pulsamos en **'Nuevo proyecto'** y seleccionamos "*Java application*".
2. Ponemos el nombre de nuestro proyecto, elegimos **dónde queremos que se guarde** y en "Create main class" lo tendremos que desmarcar. De esta forma no creará la clase principal y tendremos un proyecto vacío.
3. Una vez hecho esto, en el paquete principal (que estará vacío) pulsamos con el botón derecho y elegimos '**nuevo**' y "*JFrame form*" (esto es una ventana de interfaz gráfica).
4. Le otorgamos un **nombre a nuestra ventana**. Lo que vamos a crear es una clase que hereda de javax.swing.Jframe, por lo que el nombre de las ventanas seguirá la nomenclatura de las clases.
5. Ahora pulsamos en ejecutar el proyecto y nos aparecerá un diálogo diciendo que no hay clase principal, con la ventana que acabamos de crear. Pulsamos en 'aceptar' y ya tendremos configurado el proyecto para que ésta sea la clase que se ejecute en primer lugar, es decir, tenga interfaz gráfica por defecto.

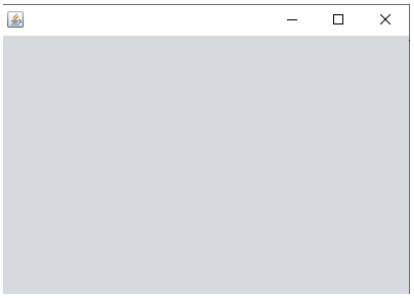

Fig 2. Proyecto vacío con interfaz gráfica.

2.1. Elementos principales del proyecto con interfaz

Cuando hemos creado nuestro proyecto con interfaz gráfica han aparecido elementos que antes no había.

En la siguiente figura podemos ver seleccionadas con diferentes colores las **diferentes partes** que vamos a encontrarnos.

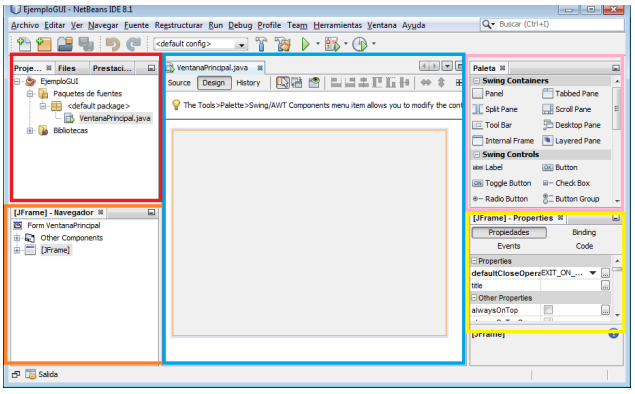

Fig. 3. Partes de un proyecto con interfaz gráfica.

- En **color azul** tenemos la vista previa de nuestra **ventana**, aquí es donde vamos a ver cómo va a ir quedando nuestra aplicación gráfica. Nos apoyaremos en las pestañas 'Fuente y Diseño'.

- En **color rojo** tenemos **el árbol de nuestro proyecto**, aquí se van a mostrar todos los paquetes, clases y bibliotecas que posea nuestra aplicación.

- En **color naranja** tenemos **el árbol de componentes del proyecto**, aquí vamos a ir viendo todos los componentes que lo componen, botones, paneles, etiquetas... Estos van a aparecer de forma jerárquica.

- En **color rosa** tenemos **la paleta de componentes**, aquí están todos los componentes de los que disponemos en *NetBeans* para agregar a nuestra aplicación. Se le pueden agregar aún más.

- En **color amarillo** tenemos **las propiedades de los elementos. C**uando pulsemos en un elemento de nuestra ventana, aquí aparecerán todas sus propiedades, texto que muestra, eventos, colores, tipo de letra, si está activo... las cuales podremos cambiar según las necesidades del desarrollo. También podremos cambiarlas en el árbol de componentes haciendo *click* derecho sobre un elemento y seleccionando 'propiedades'.

Cuando queramos escribir código que se ejecute en nuestra aplicación con interfaz gráfica, deberemos hacerlo en el constructor, justo debajo de la función *initComponents()*, la cual, inicializa todos los componentes que le agreguemos a nuestra ventana.

/ 3. Caso práctico 1: "Empezando con la interfaz"

Planteamiento: Pilar y José acaban de recibir un nuevo encargo por parte de su profesor de Programación. Este ejercicio será una primera toma de contacto con la interfaz gráfica, y consistirá en realizar un "hola mundo" utilizando los elementos que nuestros amigos consideren necesarios.

La única restricción que tiene el ejercicio es que al ejecutarse deberá mostrarse una pantalla nueva con el texto "hola mundo", mostrándolo de una forma u otra.

Nudo: ¿Cómo crees que nuestros amigos pueden realizar este ejercicio? ¿Habrá más de una forma de realizarlo?

Desenlace: Ya no es nada nuevo el decir que cuando tenemos que realizar un ejercicio en programación hay varias formas de realizarlo, siendo seguro casi todas ellas igual de correctas.

Lo primero que deberemos hacer es crear un proyecto con interfaz gráfica, que si recordamos será un proyecto de Java vacío, y deberemos crear un *JFrame* para poder insertar todos los elementos gráficos que necesitemos.

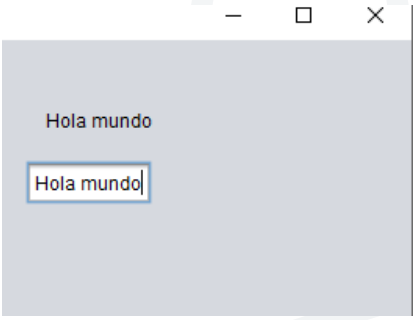

Fig. 4. Hola mundo con elementos gráficos.

En el caso que nos concierne, podemos mostrar texto de varias formas, bien utilizando una etiqueta o bien utilizando una caja de texto, de esta forma, cada uno de nuestros amigos podrá entregar el ejercicio realizado de forma diferente.

Las interfaces de usuario se deben diseñar de manera eficiente, de lo contrario no serán amigables y el uso de la aplicación será complicado para el usuario. Esto podría provocar que se pierda el interés por usar nuestros desarrollos. Por esta razón, es importante no descuidar las interfaces y hacer las pruebas pertinentes para asegurarnos de que cumplen con su objetivo principal: comunicar la aplicación con el usuario de la forma más efectiva posible.

También es importarte remarcar que, aunque podamos utilizar los dos elementos nombrados anteriormente, lo más correcto sería utilizar una etiqueta, ya que el texto de la misma no lo podremos modificar, como veremos más adelante.

/ 4. Elementos gráficos

4.1. JLabel, JTextField y JTextArea

Vamos a empezar a estudiar los principales elementos gráficos que podemos utilizar en nuestros proyectos. Se van a estudiar los componentes de texto, que se utilizarán para la entrada o salida de cadenas de caracteres:

- **JLabel:** es una *etiqueta*, y mediante ella, vamos a poder mostrar un texto cualquiera en nuestras ventanas.
- **JTextField:** es una *caja de texto*, donde vamos a poder escribir/introducir un texto cualquiera en nuestras ventanas.
- **JTextArea:** es *otro tipo de caja de texto*, donde también vamos a poder escribir/introducir un texto cualquiera en nuestras ventanas, pero, a diferencia del JTextField, éste puede tener varias líneas escritas.

Todos los elementos los vamos a poder encontrar **en la paleta de componentes**, y para poder ponernos en nuestra ventana de trabajo, basta con seleccionarlos y arrastrarlos donde necesitemos. Inmediatamente después de arrastrarlos se colocarán de forma automática donde los hayamos soltado.

Con estos elementos podremos hacer dos operaciones: **cambiarles el texto y/o obtener el texto que contienen.**

Para cambiar el texto podemos hacerlo desde la ventana de propiedades, con la **propiedad *text*.** Para obtener el texto que contienen, deberemos hacerlo **mediante código.**

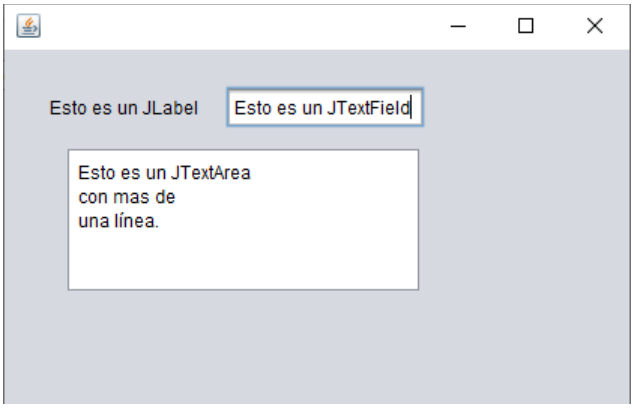

Fig. 5. Ejemplo de JLabel, JTextField y JTextArea.

Cabe destacar que todo lo que podamos hacer mediante las propiedades lo podremos hacer mediante código.

Las funciones para cambiar y obtener el texto son:

- **getText():** Esta función devuelve el texto que hay en el JLabel, JTextField o JTextArea.
- **setText(String texto)**: Esta función modifica el texto del JLabel, JTextField o JTextArea, sustituyéndolo por el que le pasemos como parámetro.

4.2. JButton

Los *JButton* son **botones que podremos poner en nuestras ventanas gráficas, y que nos permitirá interactuar con la aplicación que estemos desarrollando.** La idea es bastante simple, tenemos un botón que al pulsarlo se ejecutará una determinada acción (evento) que probablemente esté relacionada con los demás elementos que tengamos en nuestra ventana.

Los botones, al igual que todos los elementos, se encuentra **en la paleta de componentes** y para poder ponernos en la ventana de trabajo basta con seleccionarlos y arrastrarlos donde necesitemos. Igualmente, como el resto de elementos, inmediatamente después de arrastrarlos se colocarán de forma automática donde los hayamos soltado.

Con los botones podremos realizar las siguientes operaciones:

- Cambiar el texto que aparece en ellos.
- Obtener el texto que aparece en ellos.
- Hacer que realicen una función al pulsarlos.

Para cambiar y obtener el texto del botón tenemos los métodos *getText* y *setText,* al igual que con los *JLabel, JTextField* y *JTextArea*.

Para provocar que realicen una determinada acción cuando sean pulsados, tendremos que conocer el concepto de **evento.**

Un evento es un **suceso que va a tener lugar dentro de nuestro programa tras una determinada acción**: situar el cursor encima de un elemento, pulsar un elemento, escribir en un elemento, etc.

Para poder hacer que un botón ejecute una acción al pulsarlo tendremos que sobrecargar el **evento *actionPerformed,*** lo cual conseguiremos haciendo doble click sobre el botón deseado, abriéndose la ventana de código con una nueva función creada, la cual, se ejecutará cuando se pulse el botón.

En esta función podremos escribir el código que necesitemos en nuestro desarrollo y éste se ejecutará automáticamente al pulsar el botón.

Cada botón tendrá un **evento *actionPerformed*** diferente, siendo éstos independientes unos de otros. El evento se podrá consultar **en las propiedades del elemento**, en la pestaña de eventos.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // Aquí se escribirá el código que se ejecutará
    // al pulsar el botón
}
```

Fig. 6. Ejemplo de evento actionPerformed.

Vídeo 1. "Dando funcionalidad a un Jbutton"

https://bit.ly/3pJzrPQ

4.3. JCheckBox

Los *CheckBox* son los conocidos **cuadros de elección múltiple que nos podemos encontrar en cualquier programa o web de registro.**

Con estos campos podremos elegir varias opciones sobre una posible pregunta.

Estos elementos se usan principalmente en **registros de usuarios,** aunque podrán estar presentes en infinidad de situaciones y contextos.

Los CheckBox, siguen el mismo proceso de activación que el resto de elementos: selección desde la paleta de componentes, y arrastre hasta el punto donde requiera nuestro desarrollo.

Con los *CheckBox* podremos realizar las siguientes operaciones:

- Cambiar el texto que aparece en ellos.
- Obtener el texto que aparece en ellos.
- Seleccionarlo.
- Deseleccionarlo.
- Saber si está seleccionado o no.

Para cambiar y obtener el texto del botón tenemos los métodos *getText* y *setText*, al igual que con los *JLabel, JTextField* y *JTextArea*.

Estos elementos introducen las siguientes **nuevas funciones:**

- **isSelected():** Esta función devuelve verdadero o falso según el *CheckBox* esté seleccionado o no.
- s**etSelected(boolean valor):** Esta función cambia el estado del *CheckBox* a 'seleccionado' o 'no seleccionado' según el valor que se le pase.

Las acciones de cambiar texto, obtener el texto, seleccionar y deseleccionar los *CheckBox* podrán realizarse desde el panel de propiedades.

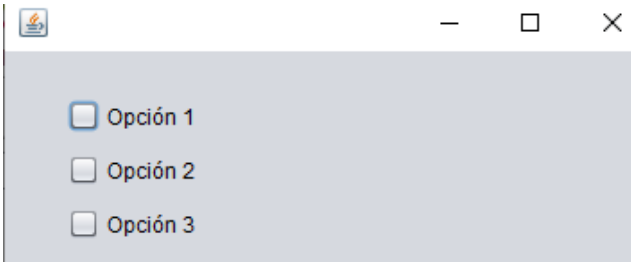

Fig. 7. Ejemplo de JCheckBox. Cada uno de ellos puede tener dos estados: marcado o desmarcado.

4.4. JRadioButton y ButtonGroup

Los *JRadioButton* son los **pequeños círculos de única elección que nos podemos encontrar en cualquier programa, web o aplicación.** Como los *CheckBox,* aunque estos elementos se usen principalmente en registros de usuarios, su utilización está extendida a casi toda aplicación que necesite **recabar información de usuarios.**

De la misma forma que el resto de elementos estudiados, la selección y arrastre de los mismos desde la paleta de componentes, provocará su activación.

Con los *JRadioButton* podremos realizar las siguientes operaciones:

- Cambiar el texto que aparece en ellos.
- Obtener el texto que aparece en ellos.
- Seleccionarlo.
- Deseleccionarlo.

Para cambiar y obtener el texto del botón tenemos los métodos *getText y setText,* al igual que con los *JLabel, JTextField y JTextArea.* Estos elementos introducen las siguientes nuevas funciones (básicamente las mismas que para CheckBox, con la diferencian en únicamente en el modo de selección):

- **isSelected():** Esta función devuelve verdadero o falso según el JRadioButton esté seleccionado o no.
- **setSelected(boolean valor)**: Esta función cambia el estado del JRadioButton ha seleccionado o no seleccionado según el valor que se le pase.

Igualmente, las acciones de cambiar texto, obtener el texto, seleccionar y deseleccionar los JRadioButton podrán hacerse desde el **panel de propiedades.**

Para hacer que solo se pueda seleccionar un único *JRadioButton* tendremos que utilizar los ***ButtonGroup.*** Una vez tengamos un grupo creado podremos introducir en él los *JRadioButton* con la propiedad button group, haciendo que de ese grupo solo pueda haber uno activo. Con los grupos podremos saber qué *JRadioButton* está seleccionado.

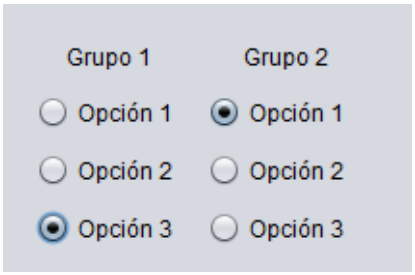

Fig. 8. Ejemplo de JRadioButton y ButtonGroup.

4.5. JComboBox

Los *JComboBox* son **las listas desplegables de varios valores que nos encontramos normalmente en los formularios webs o aplicaciones que trabajen con datos 'seleccionables'.** De estas listas se podrá elegir un valor entre los datos que ofrecen.

Estos elementos se usan de forma habitual en los formularios web dedicados a la recogida de información del usuario.

Los *JComboBox,* también pasan por la selección desde la paleta de componentes y arrastre hasta el punto deseado, para poder trabajar con ellos.

Con los JComboBox podremos realizar las siguientes operaciones:

- Agregar un elemento a la lista.
- Obtener el elemento que está seleccionado en la lista.

Para introducir valores en la lista del *JComboBox* accederemos a la pantalla de propiedades y seleccionamos la propiedad **model.** El modelo de los *JComboBox* se refiere a los datos que éstos incluyen en la lista. Aquí introduciremos todos los valores que necesitamos que aparezcan separados por un '*intro'*.

Una vez introducidos los valores, al ejecutar el programa, el *JComboBox* estará formado con todos ellos.

Para agregar un elemento utilizaremos el **método *addItem,*** al que le pasaremos un ***String*** con la línea que queremos agregar.

Para saber qué elemento se ha seleccionado deberemos usar el **evento *actionPerfomed***, que también nos servirá para saber cuándo seleccionamos un nuevo elemento. Usaremos el método getSelectedItem(). toString() del *JComboBox* para obtener el elemento seleccionado, o bien getSelectedIndex() para obtener el índice del elemento seleccionado.

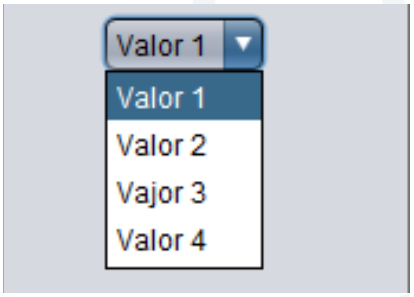

Fig. 9. Ejemplo de JComboBox.

4.6. JTable

Los *JTable* son **tablas que nos van a permitir mostrar la información en nuestras ventajas de forma ordenada, la cual, la vamos a poder dividir en filas y en columnas.**

Con los *JTable* podremos hacer las siguientes acciones:

- Insertar una fila.
- Eliminar una fila.

Para poder manipular los elementos de los JTable tendremos que utilizar su modelo, que podemos obtener de la siguiente forma:

```
DefaultTableModel modelo = (DefaultTableModel) jtable.getModel();
```

Una vez lo tengamos podremos:

- **Insertar una fila:** Para esto deberemos usar el método ***addRow*** del modelo, al que le tendremos que pasar un *array de String* con los elementos que queremos que muestre. Éstos deberán ir en el mismo orden de las columnas a mostrar.

- **Agregar una columna:** Para esto utilizaremos el método ***addColum*** del modelo, al que le pasaremos un String con el nombre de la nueva columna.

- **Eliminar una fila:** Para esto utilizaremos el método ***removeRow*** del modelo, al cual le pasaremos el índice de la fila que queremos eliminar.

- **Saber qué fila hay seleccionada:** Para esto utilizaremos el método ***getSelectedRow o getSelectedRows*** si hay más de una. El primero devolverá un entero con la fila seleccionada y el segundo un array de enteros con las filas seleccionadas.

- Saber qué columna hay seleccionada: Para esto utilizaremos el método ***getSelectedColumn o getSelectedColumns*** si hay más de una. El primero devolverá un entero con la columna seleccionada y el segundo un array de enteros con las columnas seleccionadas.

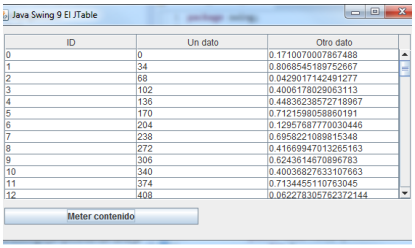

Fig. 10. Ejemplo de Jtable

Vídeo 2. "Introducir datos en un Jtable"
https://bit.ly/38cYvJg

/ 5. Layouts: Borderlayout

Ya hemos comprobado que tenemos multitud de elementos para poder agregar a las ventanas. Cuando los vamos agregando, éstos se colocan en el mismo lugar en donde los hayamos arrastrado, pero ¿qué ocurre cuando cambiamos el tamaño de la ventana?

Cuando hacemos esto, los elementos se van a quedar en el mismo sitio donde estaban, no adaptándose al nuevo tamaño de la ventana. Para solucionar esta circunstancia disponemos de los ***Layouts***, que van a hacer que esta **tarea se automatice.**

Éstos también son conocidos como ***maquetadores.***

Existen diferentes tipos de *layouts,* y cada uno va a colocar los elementos de una forma diferente, ayudándonos a crear ventanas con los elementos organizados de diferentes formas.

No obstante, previamente, tendremos que conocer un nuevo elemento, los ***JPanel.*** Estos elementos nos **permitirán colocar un panel dentro de la pantalla para agregarle elementos en su interior**, exactamente de la misma forma que hemos estado haciendo hasta ahora. Podremos colocar tantos como queramos en nuestras ventanas.

Para cambiar un *layout*, en el panel del árbol de elementos, pulsaremos con el botón derecho sobre un panel y elegiremos la opción de cambiar layout, apareciendo tras esto todos los disponibles.

El primer layout que vamos a estudiar es el ***BorderLayout.*** Este gestiona los elementos en cinco posiciones diferentes: *Norte, Sur, Derecha, Izquierda y Centro.*

En cada posición podremos poner un único elemento, pero si queremos poner más de uno podremos colocar un panel y agregar ahí todos los elementos que necesitamos, pudiendo cambiar el layout de ese panel también, haciendo un anidamiento de layouts.

Fig. 11. Ejemplo de BorderLayout.

Audio 1. "Recomendación sobre Layouts2
http://bit.ly/2Xa6qAu

/ 6. Caso práctico 2: "Mis elementos no se adaptan a la ventana"

Planteamiento: Pilar y José continúan realizando el programa de gestión de pedidos de la barra de bar. Ya se han decidido a ponerle interfaz gráfica y están creando las ventanas necesarias. "Esto es más fácil y divertido de lo que imaginaba, las ventanas se crean en un momento", le dice José a Pilar.

Pilar también está bastante contenta con la facilidad de la creación de las interfaces gráficas, pero su sorpresa llega cuando ejecutan el programa y al maximizar la primera ventana los elementos se han quedado exactamente en el mismo sitio, dejando una ventana prácticamente vacía. "¿Qué ha pasado?, si hemos creado las ventanas correctamente", se preguntan nuestros amigos.

Nudo: ¿Qué piensas al respecto? ¿Por qué crees que ha ocurrido esto?

Desenlace: Lo más normal cuando estamos aprendiendo a crear interfaces gráficas es que descuidemos uno de los pasos más importantes.

Más allá de colocar todos los elementos necesarios en su lugar correspondiente, nos solemos olvidar de que las ventanas pueden (y seguro lo harán) cambiar de tamaño.

Cuando una ventana cambia de tamaño, bien sea para ser más pequeña o más grande, los elementos de la misma se han de adaptar, si no, quedará una ventana con los elementos cortados o con gran cantidad de espacio vacío.

Para solucionar esto disponemos de los *Layouts* o maquetadores. Si usamos el *Layout* apropiado, cuando cambiemos el tamaño de nuestras ventanas observaremos que los elementos de las mismas se adaptarán automáticamente a ellas, sin tener que hacer nada al respecto.

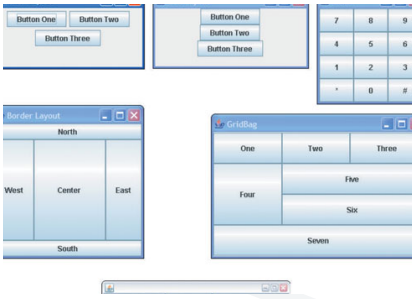

Fig 12. Distintos tipos de layouts

Al principio cuesta un poco hacerse con el funcionamiento de los *layouts,* pero con la práctica aprenderemos a identificar el tipo que necesitamos según el caso, y nuestras interfaces adquirirán una apariencia muy profesional. Merece la pena invertir tiempo en experimentar con ellos.

/ 7. Layouts: Flowlayout y Gridlayout

Otro de los layouts a tener en cuenta dado su amplio uso que seguro haremos del mismo, es el FlowLayout.

Con este layouts todos los elementos que agreguemos estarán siempre centrados según el tamaño de la ventana que tengamos.

En caso de que no quepan todos los elementos contiguos, se pondrán unos debajo de los otros para ir adaptándose.

Fig. 13. Ejemplo de FlowLayout donde los elementos se muestran flotantes sobre la ventana

El ***GridLayout***, básicamente, nos va a permitir **maquetar los elementos como si fueran una tabla.**

Un ejemplo muy sencillo para entender este layout es pensar en una calculadora. La distribución de los botones de la misma, tiene igual disposición que uno de estos *layouts*.

Cuando agregamos un *GridLayout* a un panel podremos elegir el número de filas y columnas que queramos que tenga. Para ello seleccionamos el *layout*, y en la ventana de propiedades podremos configurarlo con las propiedades *rows y columns*. También tenemos las propiedades *horizontal gap y vertical gap*, las cuales nos van a servir para definir un espacio de separación entre los elementos en vertical y horizontal.

Fig. 14. Ejemplo de GridLayout de 2 filas y 2 columnas.

/ 8. Nombre de los elementos

Ya hemos visto que tenemos multitud de elementos gráficos para poder usar en nuestras interfaces, siendo esto muy potente, debido a la gran variedad de posibilidades que dispondremos para nuestros diseños. No obstante, esto a su vez puede ser un problema.

Cuando utilicemos muchos elementos en una interfaz gráfica y queramos tratarlos, ya sea cambiándoles el texto que muestran, queriendo obtener un elemento de una lista, queriendo saber qué radio botón está seleccionado, etc., se puede convertir en una tarea difícil ya que éstos se van a ir nombrando automáticamente, y no puede que no sepamos a qué objeto tenemos que dirigirnos.

Para evitar esto, podemos **nombrarlos de una forma sencilla** que nos permita identificar rápidamente con qué elemento queremos trabajar.

Esto lo vamos a conseguir de la siguiente forma: **Como primera letra del nombre del elemento pondremos un identificativo de su tipo,** por ejemplo, para los JButton una b, e **inmediatamente después, la acción que realiza dicho elemento.**

Por ejemplo, si tenemos un *JButton* para insertar datos en una tabla podremos llamar *bInsertar*, así, rápidamente sabremos a qué elemento tenemos que dirigirnos.

Podemos seguir las siguientes reglas:

- **JLabel** - empezará pr l
- **JButton** - empezará por b
- **JTextField** - empezará por t
- **JTable** - empezará por tb
- **JRadioButton** - empezará por rb
- **JTextArea** - empezará por ta
- **JCheckBox** - empezará por ch
- **JComboBox** - empezará por cb
- **JPanel** - empezará por p
- **JTable** - empezará por tb

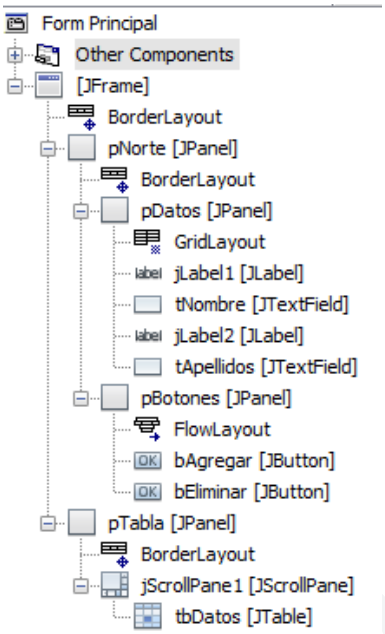

Fig. 15. Nombrado correcto de elementos gráficos

/ 9. Resumen y resolución del caso práctico de la unidad

A lo largo de esta unidad hemos aprendido cómo podemos crear un proyecto en *NetBeans* que tenga **interfaz gráfica.**

En primer lugar, hemos conocido muchos de los **elementos** que podremos utilizar, aunque no todos, como los *JTextField, los JLabel, los JButton, Jtable...*

Posteriormente, hemos visto cómo podemos hacer que un **botón** tenga una funcionalidad al pulsarlo. También, cómo hacer para que cuando utilicemos una lista de elementos, podamos saber qué elemento hay seleccionado. Otra característica estudiada, ha sido aprender a que los radio botones estén agrupados, consiguiendo así que podamos limitar a uno, la selección de los que pertenezcan a cada grupo.

Seguidamente hemos conocido cómo podemos maquetar automáticamente los elementos mediante los ***Layouts,*** habiendo diferentes de éstos, que maquetarán de forma distinta los elementos, según necesitemos. En el último punto del tema vimos una norma para **nombrar** los elementos que tengamos en nuestras ventanas, haciendo así mucho más fácil trabajar con ellos.

Resolución del caso práctico inicial

Hasta ahora todos los programas que hemos ido realizando han sido en consola, en los que simplemente se pedían los datos necesarios por teclado y se mostraban los resultados por pantalla. Uno de los aspectos más importantes en un programa es que tenga interfaz gráfica, ya que un programa complejo no tiene sentido estar ejecutándolo en consola como se hacía en los primeros tiempos de la informática. Cuando dotamos de interfaz gráfica a nuestros programas estamos dándole un toque mucho más profesional, y facilitando la tarea de los usuarios que los van a usar, ya que serán mucho más sencillos de entender y manejar.

/ 10. Bibliografía

Colaboradores de Wikipedia. (2019, octubre 22). Swing (biblioteca gráfica) - Wikipedia, la enciclopedia libre. Recuperado 30 de mayo de 2020, de https://es.wikipedia.org/wiki/Swing_(biblioteca_gr%C3%A1fica)