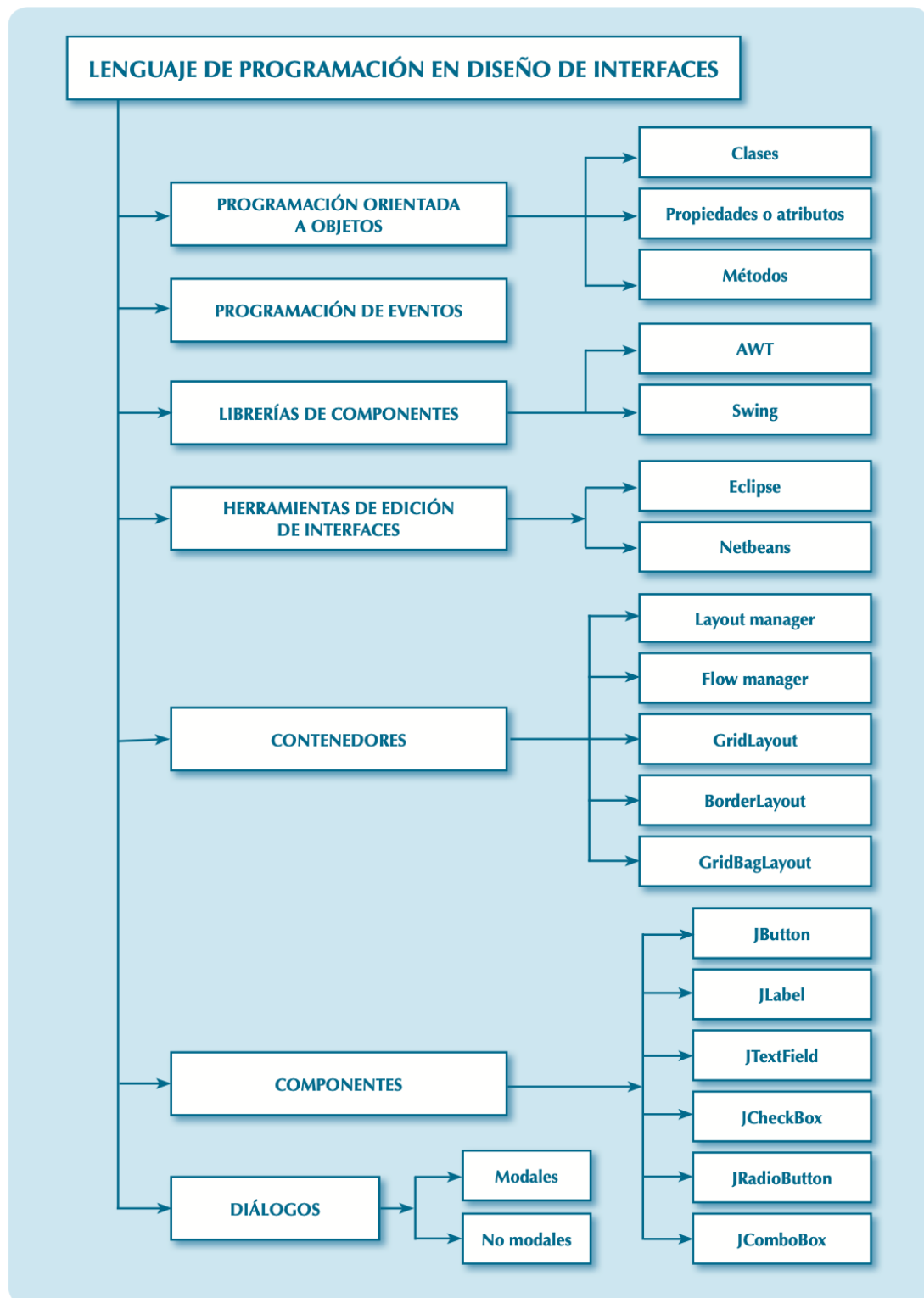


# UD2. Lenguaje de programación en diseño de interfaces

## Objetivos

- Crear una interfaz gráfica utilizando los asistentes de un editor visual.
- Utilizar las funciones del editor para ubicar los componentes del interfaz.
- Modificar las propiedades de los componentes para adecuarlas a las necesidades de la aplicación.
- Asociar a los eventos las acciones correspondientes.
- Desarrollar una aplicación que incluya el interfaz gráfico obtenido.
- Identificar las herramientas para el diseño y prueba de componentes.

## Mapa conceptual



## Glosario

**Atributo.** Estado relacionado con un objeto.

**AWT (*Abstract Window Toolkit*).** Herramientas de ventana abstracta. Biblioteca gráfica para interfaces de usuario y ventanas de Java.

**Clase.** Estructura que requiere de métodos para poder tratar a los objetos.

**Evento.** Suceso que genera la acción de un objeto.

**JFrame.** Clase utilizada de la biblioteca gráfica Swing para generar ventanas.

**Layout.** Distribución de los elementos y formas dentro de un diseño.

**Método.** Comportamiento relacionado con un objeto.

**Objeto.** Instancia de una clase. Entidad que tiene un determinado estado, comportamiento e identidad.

**Programación orientada a objetos.** Paradigma de programación en el que los objetos se utilizan como metáfora para simular entidades reales.

**Swing.** Biblioteca gráfica empleada para crear cajas de texto, botones, listas desplegables y tablas.

### 2.1. Programación orientada a objetos

El desarrollo de interfaces gráficas permite la creación del canal de comunicación entre el usuario y la aplicación, por esta razón requiere de especial atención en su diseño. En la actualidad, las herramientas de desarrollo permiten la implementación del código relativo a una interfaz a través de vistas diseño que facilitan y hacen más intuitivo el proceso de creación. La programación orientada a objetos permite utilizar entidades o componentes que tienen su propia identidad y comportamiento.

En este tema se verán en detalle los principales tipos de componentes así como sus características más importantes. La distribución de este tipo de elementos depende de los llamados *layout*, los cuales permiten colocar los elementos en un sitio o en otro.

Una misma aplicación puede presentar más de una ventana, en función de la finalidad de la misma encontramos JFrame y JDialog. La segunda establece los llamados diálogos modales o no

modales, elementos clave en el desarrollo de interfaces. La combinación de tipos de ventanas y elementos de diseño es infinita.

### 2.1.1. Clases

Una clase representa un conjunto de objetos que comparten una misma estructura (atributos) y comportamiento (métodos). A partir de una clase se podrán instanciar tantos objetos correspondientes a una misma clase como se quieran. Para ello se utilizan los constructores.

Para llevar a cabo la instanciación de una clase y así crear un nuevo objeto, se utiliza el nombre de la clase seguido de paréntesis. Un constructor es sintácticamente muy semejante a un método.

### Repaso

El constructor puede recibir argumentos, de esta forma podrá crearse más de un constructor, en función del número de argumentos que se indiquen en su definición. Aunque el constructor no haya sido definido explícitamente, en Java siempre existe un constructor por defecto que posee el nombre de la clase y no recibe ningún argumento.

### 2.1.2. Propiedades o atributos

Un objeto es una cápsula que contiene todos los datos y métodos ligados a él. La información contenida en el objeto será accesible solo a través de la ejecución de los métodos adecuados, creándose una interfaz para la comunicación con el mundo exterior.

Los atributos definen las características del objeto. Por ejemplo, si se tiene una clase círculo, sus atributos podrían ser el radio y el color, estos constituyen la estructura del objeto, que posteriormente podrá ser modelada a través de los métodos oportunos.

La estructura de una clase en Java quedaría formada por los siguientes bloques, de manera general: atributos, constructor y métodos.

### 2.1.3. Métodos

Los métodos definen el comportamiento de un objeto, esto quiere decir que toda aquella acción que se quiera realizar sobre la clase tiene que estar previamente definida en un método.

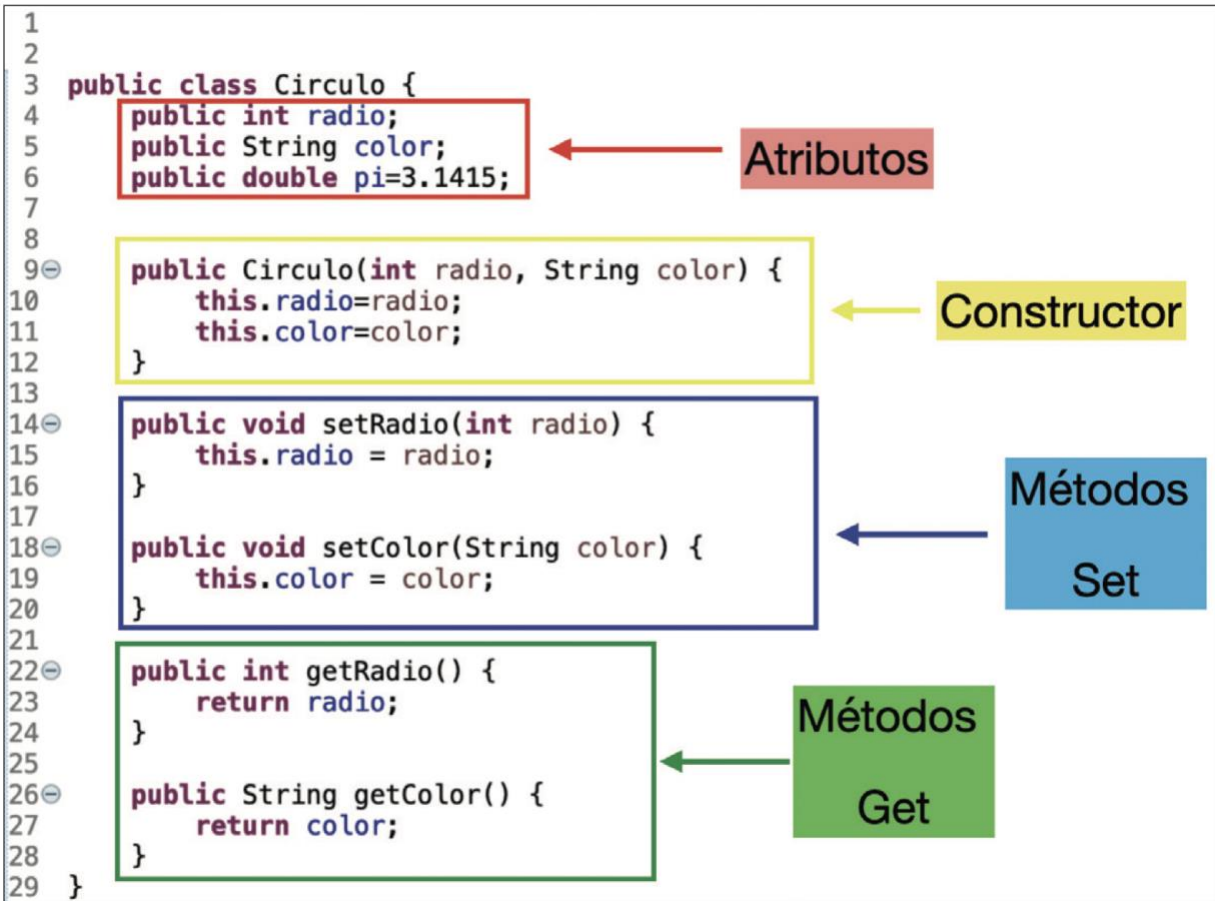


Figura 2.1 Estructura básica clase Java.

Los métodos pueden recibir o no argumentos, además, en función de su definición, devolverán un valor o realizarán alguna modificación sobre los atributos de la clase.

## 2.2. Programación de eventos

Para poder crear una conexión entre dos o más ventanas, en primer lugar, es necesario crearlas todas, ya sean de tipo `JFrame` o `JDialog`. El paso de una ventana a otra se produce tras la ocurrencia de un evento. Habitualmente, la pulsación sobre un botón.

Tras la creación de las ventanas se sitúan los botones de conexión y se modifican sus propiedades de apariencia. Este elemento puede situarse dentro de un `layout` o de un `JPanel`. Para crear el evento escuchador asociado a este botón basta con hacer doble click sobre él y de forma automática se generará el siguiente código en la clase de la ventana de la interfaz donde estamos implementando el botón conector.

Según la programación orientada a objetos, ¿qué define el comportamiento de un objeto y puede recibir o no argumentos?

```
JButton btnNewButton = new JButton("Púlsame");
btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
    }
});
panel.add(btnNewButton);
```

Figura 2.2 Evento implementado para conexión de JButton.

En el siguiente ejemplo, al pulsar el botón Jugar desde la ventana principal implementada como una clase JFrame, nos lleva a la segunda ventana de tipo JFrame, la cual muestra un mensaje en una etiqueta de texto.

Cuando se detecta la pulsación del botón como evento, desde la clase principal se crea una nueva instancia del objeto Juego, también JFrame y se especifica como visible. Finalmente, en este ejemplo, se utiliza el método dispose() el cual cierra la ventana principal y solo mantiene abierta la segunda.

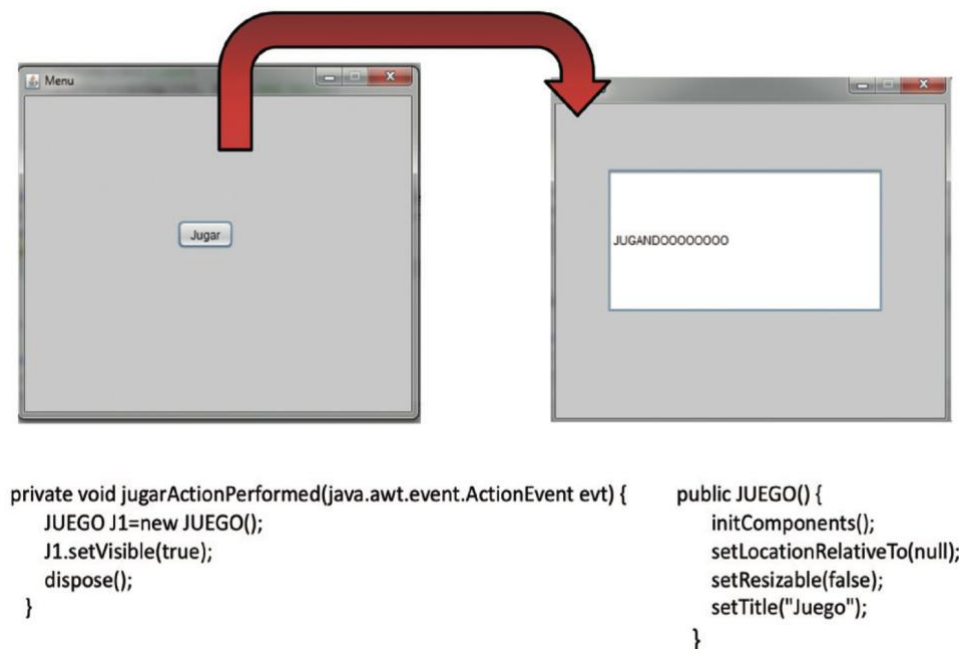


Figura 2.3 Conexión de ventanas con ActionListener.

## Nota:

Es importante tener en cuenta que siempre que se utilicen nuevas ventanas hay que ponerlas visibles utilizando el método `setVisible` (booleano de visibilidad), donde el valor que recibe por parámetro será `true` en el caso de hacerla visible y `false` en el contrario.

## Actividad 2.2

Reflexiona sobre qué tipo de aplicación puede necesitar la conexión entre ventanas. ¿Crees que es necesario utilizarlo en la mayoría de ellas o solo en algunos casos muy específicos?

## 2.3. Librerías de componentes

Algunos lenguajes de programación, entre ellos Java, utilizan las librerías, un conjunto de clases con sus propios atributos y métodos ya implementados, de esta forma pueden ser utilizados para cualquier desarrollo, reutilizando su código y consiguiendo una elevada reducción del tiempo de programación. En cuanto al desarrollo de interfaces gráficas, se requiere del uso de librerías que permiten el desarrollo de estas interfaces. En Java se distingue entre la librería AWT y Swing.

### 2.3.1. AWT

Para poder utilizar los métodos y atributos de estas clases es necesario importar las librerías en Java, para lo que se utiliza la palabra clave *import*, seguida del nombre de la librería, en concreto, de la ruta del paquete que se va a agregar. Esta importación se realiza justo después de la declaración del paquete, si esta existe.

```
import javax.swing.*;  
import java.awt.*;
```

**Figura 2.4** Código para importar librerías Swing y AWT.

En primer lugar, se desarrolló AWT (*Abstract Window Toolkit*), librería que permite, a través de la importación del paquete `java.awt`, la creación de interfaces gráficas. Dos de sus funcionalidades más importantes son el uso de la clase `Component` y de la clase `Container`. La primera define los controles principales que se sitúan dentro del elemento `container` o contenedor, la última hace referencia a la pantalla en la que se muestra la interfaz de aplicación que se va a desarrollar.

### 2.3.2. Swing

En la actualidad la librería Swing supone la evolución de la anterior, eliminando algunas limitaciones que esta presentaba, como el uso de barras de desplazamiento. Swing incorpora múltiples herramientas, métodos y componentes que permiten diseñar cualquier tipo de interfaz. A través de su entorno de diseño permite crear un nuevo desarrollo desde cero arrastrando los componentes desde la paleta de diseño, mientras que se va generando el código asociado. Conocer el funcionamiento de ambas vistas, diseño y código, permite adaptar el funcionamiento a las especificaciones de la aplicación diseñada.

	A W T	S W I N G
Usa componentes del S.O	✓	✗
Dibuja sus propios componentes	✗	✓
El S.O maneja los eventos	✓	✗
Java maneja los eventos	✗	✓
La apariencia cambia con el S.O	✓	✗
Tienen la misma apariencia en cualquier S.O	✗	✓
La apariencia es estática	✓	✗
Se pueden personalizar	✗	✓

Figura 2.5 Comparativa de la librería AWT y Swing.

### Actividad 2.3

Implementa la ventana de una interfaz con componentes gráficos utilizando la librería AWT y Swing. ¿Qué diferencias has encontrado a la hora de utilizar ambas? ¿Es más práctico el uso de la vista de diseño con paleta o la vista de código? ¿En qué casos es mejor utilizar una u otra?

### 2.4. Herramientas de edición de interfaces

Tal y como se vio en el capítulo anterior, existen varias herramientas basadas en componentes visuales para el desarrollo de interfaces. En este caso, se ha escogido la herramienta Eclipse, en base a las características expuestas en el apartado 1.3.2.



## Recurso web

Para realizar la descarga de Eclipse solo se necesita acceder al sitio web (<https://www.eclipse.org>) y escoger la versión que más se adecua al equipo en el que se va a realizar el desarrollo, el proceso de instalación solo dura unos pocos minutos.

Para que el entorno de desarrollo Eclipse funcione correctamente es necesario instalar el JDK correspondiente, Java Development Kit. La descarga de este se lleva a cabo desde la página web de Oracle. La ejecución de Eclipse es sencilla, bastará con lanzar la aplicación a través de su icono que normalmente podemos identificar bajo el nombre *Eclipse Installer*.

Una vez que se haya completado este proceso, ya tendríamos instalado todo el entorno básico para el desarrollo de interfaces posteriores. Para ejecutar Eclipse basta con pulsar sobre el icono de la aplicación, normalmente con el nombre de Eclipse Installer. Finalmente, selecciona Eclipse IDE for Enterprise Java Developers, pulsa Install y luego Launch.

## 2.5. Contenedores

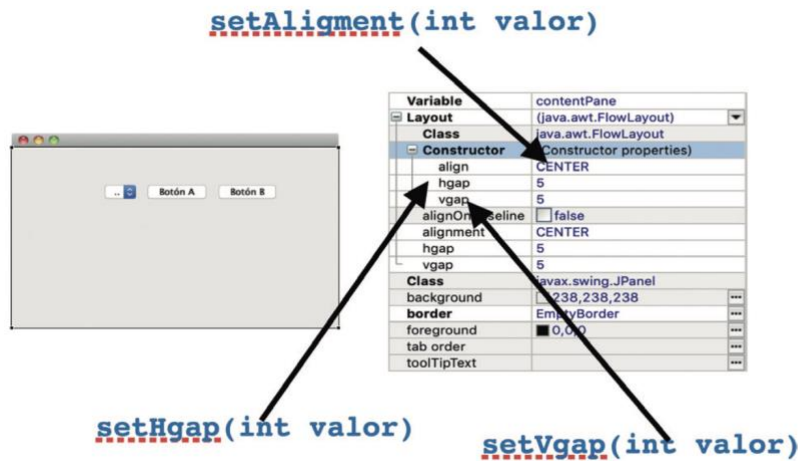
El uso de contenedores permite implementar un tipo de componente que puede contener otros componentes. Esto resulta especialmente útil para el diseño de interfaces, puesto que permite determinar la distribución y posición exacta de cada uno de sus elementos.

### 2.5.1. Layout manager

Un layout manager (manejador de composición) permite adaptar la distribución de los componentes sobre un contenedor, es decir, son los encargados de colocar los componentes de una interfaz de usuario en el punto deseado y con el tamaño preciso. Sin los layout los elementos se colocan y, por defecto, ocupan todo el contenedor. El uso de los layout nos permite modificar el tamaño de los componentes y su posición. En este apartado analizaremos el funcionamiento de los distintos layout disponibles.

## 2.5.2. FlowLayout

FlowLayout sitúa los elementos uno al lado del otro, en una misma fila. Permite dar valor al tipo de alineación (`setAlignment`), así como la distancia de separación que queda entre los elementos, en vertical (`setVgap`) y en horizontal (`setHgap`).



**Figura 2.6**  
Propiedades FlowLayout.

Figura 2.6 Propiedades FlowLayout.

## 2.5.3. GridLayout

Este layout permite colocar los componentes de una interfaz siguiendo un patrón de columnas y filas, simulando una rejilla.

Al igual que en el caso anterior, es posible modificar el valor de la separación entre componentes. Las propiedades de este elemento incorporan los atributos `cols` y `rows`, que definen el número exacto de columnas y filas. Para la creación de este sistema de rejilla se utiliza un constructor que recibe por parámetro el valor exacto de filas y columnas que tendría la interfaz, `GridLayout (int numFilas, int numCol)`.

Cualquiera de los elementos layout presentan como propiedad común el valor de `vgap` y `hgap`, que definen la distancia entre elementos que se crea tanto en vertical como en horizontal.

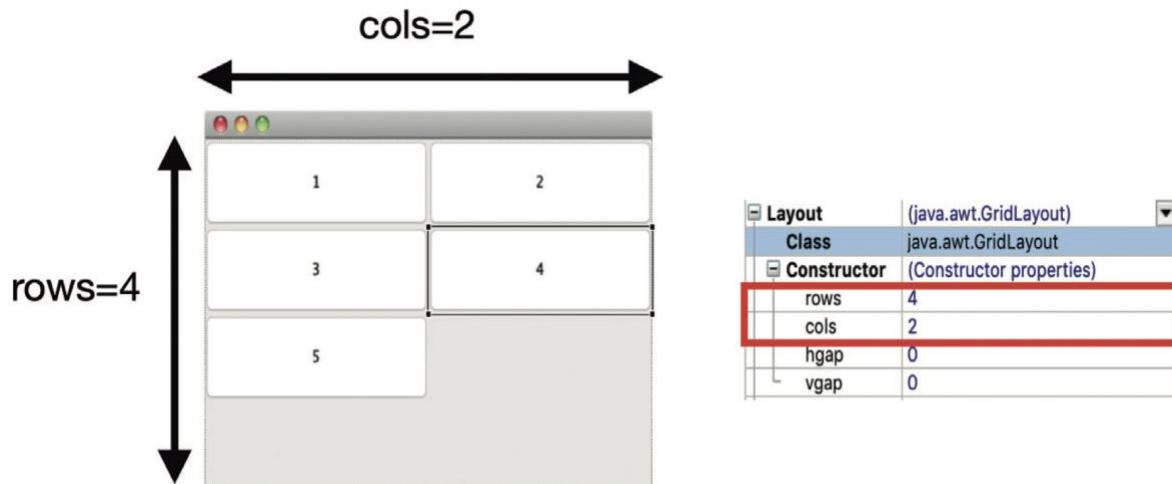


Figura 2.7 Propiedades y ejemplo de botones en GridLayout.

## 2.5.4. BorderLayout

BorderLayout permite colocar los elementos en los extremos del panel contenedor y en el centro. Para situar a cada uno de los elementos desde la vista de diseño basta con colocarlos en la posición deseada.

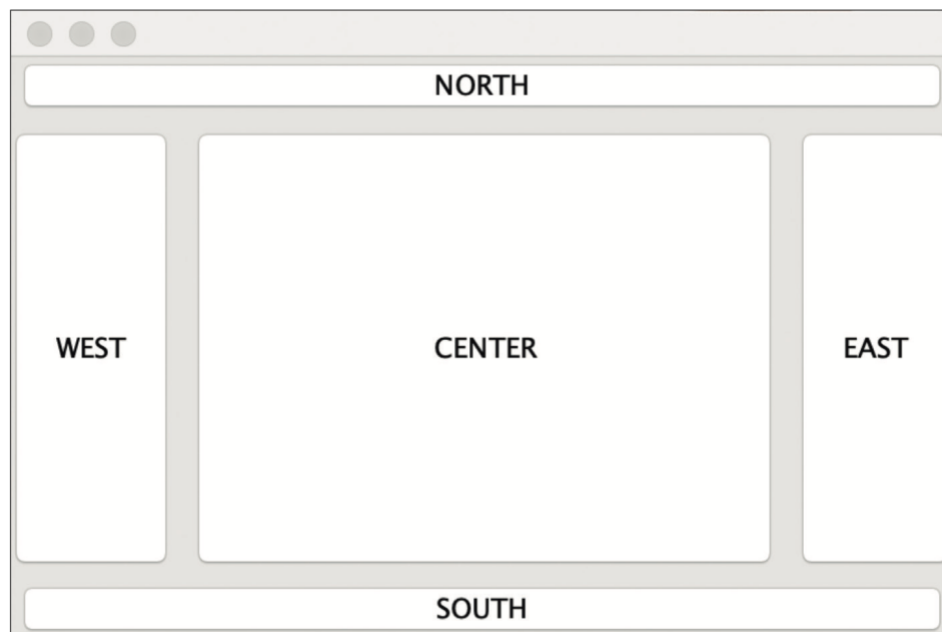


Figura 2.8 Propiedades y ejemplo de botones en BorderLayout.

Ahora bien, desde el código se sitúan atendiendo a su situación (north, south, east, west, center).

### 2.5.5. GridBagLayout

A diferencia del tipo GridLayout visto, este permite un diseño más flexible, donde cada uno de los componentes que se coloquen tendrá asociado un objeto tipo GridBagConstraints.

Tras la inserción de este layout será posible ubicar el elemento de una forma mucho más precisa, seleccionando la posición exacta de la rejilla, por ejemplo, en este caso se situará en la columna 2 y fila 2.

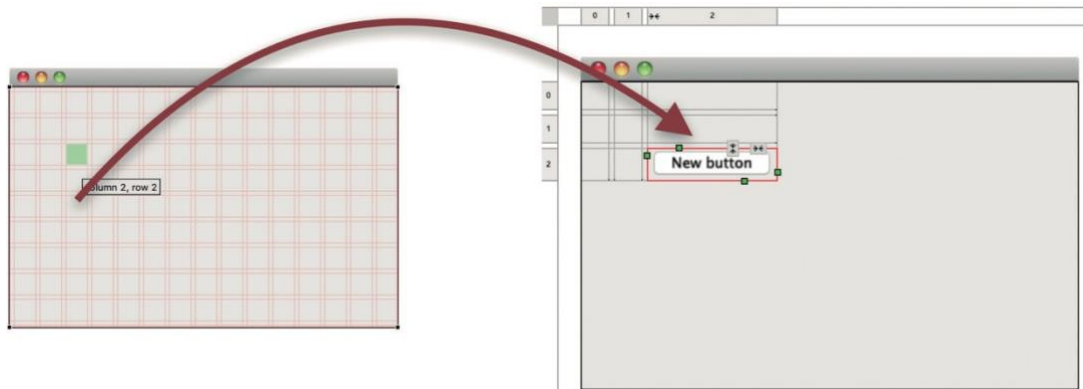


Figura 2.9 Ejemplo botones en GridBagLayout.

## 2.6. Componentes

Existe un amplio abanico de componentes, en este apartado se verán algunos de los más utilizados y que constituyen gran parte de la interfaz de cualquier entorno de desarrollo.

### 2.6.1. JButton

Permite crear un objeto de tipo botón dentro de una interfaz gráfica en Java. Las propiedades de este elemento permiten modificar varios aspectos relativos a su apariencia:

#### Propiedades JButton

### 2.6.2. JLabel

Este elemento es uno de los más sencillos de aplicar y que, al mismo tiempo, más utilidad reporta. No solo se trata de un elemento de texto, sino que este contenedor puede llegar a albergar imágenes, iconos o texto. Sus propiedades características son:

## Cuadro 2.2 Propiedades JLabel

Hay que prestar especial atención a los valores background y foreground: ambos definen el color del texto.

### 2.6.3. JTextField

El elemento JTextField se utiliza como contenedor de una línea de texto; el tamaño queda definido por el valor del atributo “columns”. No se trata de un valor exacto en cuanto a número de caracteres, sino que está definiendo su ancho, por lo tanto, en función del carácter que se escriba, variará la capacidad.

## Propiedades JTextField

### 2.6.4. JCheckBox

Los elementos de tipo casilla o CheckBox son elementos que se presentan junto a una pequeña caja cuadrada y que pueden ser marcados por el usuario.

Presenta unas propiedades similares a los casos anteriores, añadiendo algunos nuevos atributos como “selected”, el cual puede ser de valor true o false: el primero indicará que la casilla se muestre marcada por defecto y si es false aparecerá sin marcar.

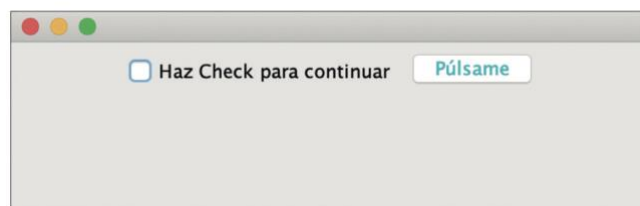


Figura 2.10 Interfaz ejemplo con JCheckBox y JButton.

### 2.6.5. JRadioButton

Los elementos de tipo JRadioButton se utilizan habitualmente en el desarrollo de interfaces para indicar varias opciones, de las que solo podrá escoger una, es decir, resultarán excluyentes. Las propiedades que presenta son iguales a la del elemento JCheckBox.

Ahora bien, cuando insertamos un elemento JRadioButton en una interfaz, su funcionamiento va a ser muy parecido a un elemento de tipo check. Para conseguir un comportamiento excluyente es necesario utilizar un objeto tipo ButtonGroup.

La creación de un elemento `ButtonGroup` nos permite asociar a este grupo tantos elementos como se deseen, de esta forma todos aquellos que queden agrupados resultarán excluyentes entre sí puesto que pertenecen al mismo grupo.

#### 2.6.6. JComboBox

Finalmente, otro elemento común en la creación de interfaces son los menús desplegables, los cuales se crean a través del componente `JComboBox`. Presenta unas propiedades muy parecidas al resto de componentes descritos.

Para insertar los valores que se mostrarán en el combo utilizando la vista de diseño, desde propiedades seleccionamos “model” y se abrirá una nueva ventana en la que se escribe en líneas separadas los valores del combo. El valor máximo de elementos mostrados en el combo queda establecido en la propiedad `maximumRowCount`.

La propiedad `selectedIndex` permite al desarrollador indicar cuál es el valor que mostraría por defecto, de entre todos los recogidos, siendo 0 la primera posición.

### 2.7. Diálogos

Las aplicaciones que solo utilizan una pantalla implementarán su interfaz solo con un elemento `JFrame`, pero cuando la herramienta que se está desarrollando presenta más de una ventana, las de tipo secundario se crearán utilizando `JDialog`, puesto que esta sí permite tener un elemento padre, es decir, un elemento principal a partir del cual se accede a la ventana secundaria.

Las ventanas tipo `JDialog` siempre quedarán situadas por encima de su padres, ya sea de tipo `JDialog` o `JFrame`.

La creación de este tipo de ventanas se realiza de forma similar a la de tipo `JFrame`, desde el menú `File` y `New` seleccionamos `Other` y, a continuación, dentro de la carpeta `WindowBuilder` pulsamos sobre `JDialog`.

En la figura 2.11 se muestra el resultado que se genera al crear un `JDialog` de la forma descrita. En este primer diseño aparecen dos botones, `Ok` y `Cancel`.



Figura 2.11 Ventana creada JDialog.

- a) *Diálogos modales*: son aquellos que no permiten que otras ventanas de diálogo se abran hasta que la que se encuentra abierta no se haya cerrado, por ejemplo, un programa que queda a la espera de la selección de una opción para poder continuar, como la selección del número de asiento en una aplicación para la compra de billetes de tren.
- b) *Diálogos no modales*: sí permitirán que haya tantos JDialog abiertos como se deseen.

Para indicar a cuál de estos tipos pertenecen utilizamos el flag de modal del constructor de JDialog, indicando a true para modal y false para no modal.

```
JDialog ventanaSec = new JDialog(f, "Dialog", true);
```

Figura 2.12 Creación de un elemento JDialog de tipo modal.

## Actividades propuestas

4. **2.4.** ¿En qué casos crees que es más conveniente que las aplicaciones utilicen JDialog de tipo modal o no modal?
5. **2.5.** Indica si el siguiente código es correcto en el caso de querer utilizar un diálogo de tipo no modal:
6. `JDialog ventanaSec = new JDialog(f, "Dialog", false);`

## Resumen

n El uso de interfaces gráficas da lugar a una comunicación entre usuario y aplicación que se basa en un diseño accesible y útil. Actualmente, existen en el mercado herramientas de desarrollo como

*Eclipse*, que permite la implementación del código o el uso de vistas de diseño para el proceso de diseño y creación de interfaces.

n El paradigma de la *programación orientada a objetos* permite manipular objetos como metáfora para simular entidades reales. Estas instancias son entidades que tienen un determinado estado, comportamiento e identidad.

n Una interfaz puede presentar más de una ventana; en función de la finalidad de la misma encontramos *JFrame* y *JDialog*. La segunda establece los llamados diálogos modales o no modales, elementos clave en el desarrollo de interfaces. Dentro de una interfaz, la distribución de los elementos se establece utilizando disposiciones de tipo layout, los cuales permiten colocar los elementos en un sitio o en otro.

n La librería *Swing* sirve para programar todo tipo de componentes visuales como botones, etiquetas, menús desplegables o casillas de verificación, entre muchos otros. Por otro lado, *Swing* es una extensión para *AWT*, un kit de herramientas de widgets utilizados para el desarrollo de interfaces gráficas en Java.

n Aunque el número de elementos que se incorporan en la paleta (*Palette*) de estas librerías es muy amplio, en este capítulo se han descrito algunos de los más usuales, analizando sus principales propiedades. El repertorio de elementos disponibles permite crear infinitas combinaciones que se adecuarán en cada caso a las especificaciones finales de la aplicación. Algunos de los elementos más importantes son: