DESARROLLO DE INTERFACES

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Introducción a la confección de interfaces

Un **lenguaje de programación** consiste en un conjunto de reglas y normas que permiten a una persona (en este caso un programador) escribir un conjunto de instrucciones interpretables por un ordenador, cuyo **objetivo** es **controlar** diferentes **comportamientos lógicos o físicos de una máquina**.

De manera tradicional, se ha establecido una clasificación entre lenguajes de bajo y alto nivel. Los primeros se encuentran más cerca de lo que es capaz de entender un ordenador, ejercen un control directo sobre el hardware y están más alejados de la lógica humana (lenguaje máquina con 0 y 1 o lenguaje ensamblador).

Los anteriores resultan muy difíciles de entender por una persona. Por esa razón, aparecen los lenguajes de alto nivel, los cuales pueden ser descritos utilizando reglas comprensibles por el programador, con un lenguaje más cercano al natural. Será durante el proceso de compilación del código fuente de los programas, cuando estos se traduzcan a un lenguaje de bajo nivel, capaz de ser entendido por una máquina.

Ahora bien, las herramientas desarrolladas a través de un lenguaje de programación, sea del tipo que sea, **requieren del desarrollo de una interfaz** que permita una interacción con el usuario de la misma, de lo contrario, se requeriría que todos fuéramos programadores expertos para poder utilizar cualquier aplicación atendiendo a su lenguaje fuente.

Paradigmas de programación

Un paradigma de programación define un **estilo de programación**. Los paradigmas describen la **estructura del programa** que va a dar solución a los problemas computacionales.

En primer lugar, encontramos el **modelo imperativo**, que consiste en un conjunto de instrucciones ordenadas de **forma secuencial y claramente definidas** para su ejecución en una máquina, es decir, definen un paso a paso. Este modelo se divide en **otros tipos**:

- **Programación estructurada:** Incluye **estructuras de control** que permiten evaluar los casos para decidir entre un camino de instrucciones u otro. También se incorporan estructuras **iterativas**.
- **Programación procedimental o basada en funciones:** Subdivide en **subrutinas** y funciones de menor tamaño que **simplifican** la programación, aligerando su implementación y posterior mantenimiento.
- **Programación modular:** Finalmente, este tipo de programación permite desarrollar cada **programa** de forma completamente **independiente** al resto del código, lo que agiliza las tareas de implementación y prueba. Será en la parte final del proceso cuando **se combinen todos** los módulos, creando el software definitivo.

Algunos lenguajes conocidos que utilizan la **programación imperativa** son: Java, C, C#, Python o Ruby.

En el modelo imperativo se indica la **secuencia de pasos exacta** que se ha de seguir para resolver un problema.

Por el contrario, en el caso del **modelo declarativo**, **no se describen los pasos**, sino el **problema** que se plantea.
Algunos ejemplos de lenguajes descriptivos son HTML, CSS y SQL.

Programación orientada a objetos, eventos y componentes

Encontramos otros modelos de programación que incluyen características propias de los definidos en el apartado anterior. Es el caso de la programación orientada a objetos, eventos o componentes.

La combinación de estos tres tipos resulta **clave para el desarrollo de interfaces** que veremos en este módulo.

- **Modelo orientado a objetos:** El funcionamiento de este tipo de programas se basa en la creación de **entidades**, que reciben el nombre de objetos, las cuales tienen asociados **atributos, propiedades y métodos**. La interacción entre los objetos permite resolver los problemas de computación planteados. Algunos lenguajes de programación orientados a objetos son: Java, Ruby, Visual Basic, Perl, PHP o Python, entre otros.
- **Modelo basado en eventos:** Este modelo es uno de los más recientes. Su funcionamiento viene determinado por **acciones** externas, por ejemplo, la pulsación sobre un botón. Uno de los lenguajes típicos de este tipo de programación es **JavaScript**, que utiliza manejadores de eventos, tanto en el lado del cliente como del servidor (Node.js).
- **Modelo basado en componentes:** La clave de este último modelo es la **reutilización de módulos** de software desarrollados previamente.

Para llevar a cabo esta tarea la mayoría de los entornos de desarrollo integrado (IDE) permiten desarrollar componentes visuales, permitiendo empaquetar el código para reutilizarlo posteriormente.

Herramientas propietarias y libres de edición de interfaces

La motivación principal para utilizar herramientas de desarrollo software basado en componentes visuales radica en que, una vez empaquetados, se podrán compartir con otros desarrolladores y, por tanto, serán reutilizables. Esto supone un mayor ciclo de vida que el desarrollo tradicional por comandos. Si no se desarrolla utilizando componentes y se realiza de manera directa, se producirá un incremento de tiempo y costes asociados al proyecto.

Destacamos a continuación las principales herramientas de desarrollo software:

NOMBRE	LICENCIA	LENGUAJES SOPORTADOS	ENLACE
Visual Studio *Community	Propietaria *Libre	C#, HTML, Javascript, XML	https://visualstudio.microsoft.com/es/
Mono Develop	Libre	C#, Java, .NET, Python	https://www.monodevelop.com
Glade	Libre	C++, C#, Java, Python	https://glade.gnome.org
NetBeans	Libre	Java, HTML, PHP, Python	https://netbeans.org
Eclipse	Libre	Java, C++, PHP	https://www.eclipse.org

Visual Studio

Entre las fortalezas más importantes de este IDE se encuentra el **uso de lenguajes multiplataforma**. Se puede escribir en los lenguajes C#, F#, Razor, HTML5, CSS, JavaScript, Typescript, XAML y XML.

Además, este entorno de desarrollo incorpora la funcionalidad de autocompletado de código mientras estamos programando, por lo que es fácil detectar los problemas en **tiempo real**, ya que nos recuerdan que es posible que se esté cometiendo algún fallo a través de líneas rojas y onduladas que se muestran en el editor bajo una línea o fragmento de código. A la hora de depurar el código, permite ir paso a paso, estableciendo puntos de interrupción, por procedimientos y por instrucciones.

Por último, cabe destacar que permite **administrar el código en los repositorios más utilizados** en la actualidad como son GIT, GitHub y Azure DevOps. De esta forma, es posible controlar las modificaciones entre las diferentes versiones de nuestros proyectos y poder realizar copias de seguridad de los archivos durante el desarrollo del mismo.

MonoDevelop

Este IDE libre y gratuito proporciona las funcionalidades propias de un **editor de texto**, además de las propias de un entorno para **depurar y gestionar proyectos.**

Entre sus principales ventajas encontramos que permite trabajar con algunos de los lenguajes más demandados en la actualidad, como son C#, Java, .NET y Python. Pertenece a **Unity**, motor de videojuegos multiplataforma por excelencia. Esta plataforma es muy interesante porque permite desarrollar tanto para Windows, Mac Os X y Linux.

Glade

Este programa ayuda a la creación de interfaces gráficas de usuario y es muy utilizada en entornos **XML**. También permite el **desarrollo de interfaces gráficas basadas en lenguaje C, C++, C#, Java, Python…**

Dado que la interfaz es bastante intuitiva, se puede aprender rápido y obtener un dominio de la herramienta invirtiendo poco tiempo. La principal diferencia respecto a las demás propuestas es que está diseñada pensando especialmente en **GNU/Linux**.

NetBeans

NetBeans es una **herramienta gratuita y de código abierto.** Al igual que Eclipse, que veremos en el siguiente apartado, es uno de los entornos de desarrollo más utilizados para el desarrollo de interfaces a través lenguaje Java, aunque también permite utilizar otros lenguajes como PHP o Python.

Este IDE permite extender el entorno con un gran número de módulos que agrupan clases de Java y que permiten interactuar con las APIs de NetBeans.

Eclipse

Este IDE es de **código abierto y multiplataforma**. Dispone de la funcionalidad Graphical Layout, que nos permite visualizar el contenido en vista de diseño y desarrollar componentes visuales de una forma rápida e intuitiva. Cabe destacar su componente 'Palette', un panel que permite crear botones, cuadros de texto, cuadrículas, insertar imágenes, etc.

Para completar esta asignatura vamos a utilizar el entorno de desarrollo Eclipse, puesto que, en la actualidad, su uso es cada vez más frecuente en lo que respecta al desarrollo de interfaces de forma profesional.

Para poder obtenerlo, desde el sitio web de Eclipse primero se selecciona la versión que corresponda con el equipo con el que estemos trabajando, se descarga y, a continuación, se instala a través de unos sencillos pasos. Este proceso ocupa pocos minutos. Ahora bien, también se debe tener instalado Java Development Kit (JDK) correspondiente al sistema operativo del equipo, cuya descarga puede realizarse desde la página web de Oracle.

Una vez completado este proceso, ya tendríamos instalado todo el entorno básico para el desarrollo de interfaces posterior. 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', luego pulsa 'Install' y posteriormente 'Launch'.

Librerías
AWT y Swing

Algunos lenguajes de programación (entre ellos Java) utilizan librerías, un **conjunto de clases** con sus propios atributos y métodos ya implementados. De esta forma, pueden utilizarse posteriormente para cualquier desarrollo reutilizando su código, lo cual **reduce considerablemente el tiempo de programación**. En cuanto al desarrollo de interfaces gráficas, para poder implementarlas debemos usar librerías que permiten el desarrollo de las mismas. En Java se distingue entre la librería AWT y Swing.

Para poder utilizar los métodos y atributos de estas clases, es necesario **importar** las librerías en Java. Para ello, se utiliza la palabra clave import, seguida del nombre de la librería requerida, 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.*;
```

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

	AWT	SWING
Usa componentes del S.O.	sí	no
Dibuja sus propios componentes	no	sí
El S.O. maneja los eventos	sí	no
Java maneja los eventos	no	sí
La apariencia cambia con el S.O.	sí	no
La apariencia es estática	sí	no
Se pueden personalizar	no	sí

En la actualidad, la librería **Swing** supone la **evolución** de la anterior, eliminando algunas limitaciones que presentaba AWT (como el uso de barras de desplazamiento). Swing incorpora múltiples herramientas, métodos y componentes que permiten diseñar cualquier tipo de interfaz. A través de un entorno gráfico, permite crear un diseño desde cero arrastrando los componentes hasta la paleta de diseño, mientras que al mismo tiempo se va generando el código asociado. Conocer el funcionamiento de ambas **vistas, diseño y código**, permite **adaptar el funcionamiento a las especificaciones** de aplicación buscadas.

Instalación de Swing en Eclipse

Para la implementación de interfaces gráficas en Java, se va a utilizar la librería Swing, un kit de herramientas que permite desarrollar interfaces gráficas de usuario para programar Java. Esta librería, a diferencia de su antecesora, **garantiza** que el **diseño y comportamiento** de las aplicaciones será exactamente **el mismo**, **independientemente** del **sistema operativo**. Esto se debe a que **AWT** utiliza los **controles nativos** del sistema operativo en el que se encuentra.

A diferencia de la librería AWT, Swing proporciona una apariencia que **puede emular** varias plataformas y utilizar **componentes visuales más avanzados**. El proceso de instalación y configuración de esta librería en el entorno de desarrollo de Eclipse se describe a continuación:

- **Desde el menú Help seleccionamos Install New Software.**
- **En el menú desplegable de la parte superior de la nueva pantalla se selecciona la versión del entorno que estemos utilizando.**

En ese momento aparecerán todos los paquetes disponibles para esa versión y seleccionaremos todos los paquetes que comienzan por **SWT y WindowBuilder.** Recuerda que se encuentran dentro de la carpeta **General Purpose Tools.** Después, pulsamos continuar.

Primera clase con Java Swing. JFrame

La importación de la librería Swing de Java se realiza usando la sentencia, **import javax.swing**. De esta forma, todas las clases contenidas en el paquete serán importadas. En el caso de querer importar solo una de ellas, basta con indicarlo tras la palabra swing. Por ejemplo, si se va a utilizar la clase que crea los botones, se usaría **import javax.swing.JButton.**

Una de las clases más importantes del paquete Swing es **JFrame**, puesto que se encarga de crear las **ventanas** sobre las que se añaden el resto de elementos. La llamada a esta clase conllevará la **aparición de la vista de diseño** (Design).

En algunas ocasiones, se puede confundir la clase JFrame con la clase **JPanel**, pero mientras que la primera define una ventana completa, la segunda es solo un **contenedor de componentes**, por lo que **dentro de un Jframe podríamos encontrar múltiples JPanel (en android es el equivalente a Layout)**.

Se puede crear una clase Jframe de la manera habitual, creando una clase en Java y, a continuación, importando manualmente el paquete Swing y codificando los métodos relativos a JFrame y el resto de componentes. El resultado sería exactamente el mismo, pero se recomienda realizarlo usando la creación de clase con JFrame que se describe a continuación, puesto que, de lo contrario, no se tendría acceso a la vista de diseño que se verá en el apartado 9. La **creación** de nuestra primera **clase JFrame** se realiza en dos sencillos pasos:

- Desde **File** o pulsando con el botón derecho sobre el nombre del proyecto que se está desarrollando, seleccionamos **New.**

  Se despliega un menú con diferentes opciones, habitualmente las que se han usado más frecuentemente. Si no aparece JFrame, seleccionaremos **Other.**

• La clase JFrame se encuentra dentro de las carpetas **WindowBuilder y Swing Designer**. Seleccionamos **JFrame** y pulsamos **Next.** Finalmente, se solicita el nombre y se pulsa Finish.

Análisis del entorno de diseño en Eclipse

El área de diseño para desarrolladores de Java en Eclipse permite **desarrollar interfaces añadiendo componentes gráficos directamente**, sin necesidad de programar líneas de código. Para ello, es importante conocer la distribución de la interfaz de trabajo de este IDE.

La vista de diseño nos permite insertar cualquier tipo de elemento en la interfaz sin necesidad de escribir el código, ya que este se genera automáticamente y se puede consultar desde la pestaña 'Source'. Es decir, desde 'Design' es posible añadir todos los elementos que se quieran incluir en la aplicación. Así definiremos la parte visual y, desde el apartado dedicado al código, se cargarán los métodos relativos a cada uno de los objetos insertados, sobre los cuales podremos incluir el resto del código necesario para definir su comportamiento exacto.

A continuación, se describen los diferentes **grupos de herramientas** que podemos encontrar en Eclipse, tanto los de tipo general, necesarios para la programación de cualquier aplicación con Java, como los específicos del área de diseño.

Toolbar

En la barra de herramientas o 'Toolbar' de Eclipse se encuentran los iconos relativos a las acciones genéricas de programación, como la creación de paquetes, clases... Uno de los botones más importantes es el encargado de la ejecución del programa. Al hacer clic sobre la flecha que se encuentra a su derecha, se podrá seleccionar cuál de entre las clases Java del proyecto actual se quiere ejecutar. En el desplegable que aparece como "Run As" se podrá seleccionar el tipo de ejecución como Aplicación de Java.

Vista de diseño
General

La zona de diseño es la ventana principal del entorno de desarrollo con **WindowBuilder**, puesto que es la zona en la que se colocan los elementos de la interfaz y, además, donde se encuentran todos los componentes y características de diseño necesarios para el desarrollo de una interfaz gráfica. En la zona de la derecha de la pantalla se muestra una previsualización de la aplicación que se está implementando, podríamos decir que es el lienzo sobre el qué dibujar la interfaz.

Vista de diseño
Palette

En la parte de la izquierda del área de diseño aparece el menú **Palette**, que recoge todos los componentes, propiedades y contenedores que se utilizan en la creación de una interfaz gráfica. Desde el menú Palette se realiza todo el diseño de la interfaz, ya que incorpora múltiples herramientas como los **containers,** que definen el tipo de contenedor donde se ubican los componentes; **layouts**, que determinan la distribución exacta de los elementos; o los componentes (**components**) que queramos utilizar en la interfaz (etiquetas, campos de texto o botones, entre otros).

Los componentes gráficos son los elementos que permiten al usuario interaccionar con la aplicación. Cada uno de los componentes corresponde con una clase en Java, es decir, cada componente tendrá sus propios métodos y atributos, por lo que, cuando se quieran utilizar, bastará con crear una instancia del objeto deseado.
Para insertarlos en la zona de diseño basta con hacer clic sobre el componente y arrastrarlo hasta la zona del contenedor exacta en la que se va a ubicar.

Vista de diseño
Structure

La última sección de la vista de diseño en Eclipse recibe el nombre de **Structure** y está formada por dos partes claramente diferenciadas: components y propierties.

- **Components:** Esta zona muestra un **resumen** de todos los componentes que se han colocado en el diseño de la interfaz, como si de un **explorador de carpetas** y archivos se tratase, pero en este caso nos muestra los elementos de diseño. Como se puede observar, aparece el **nombre de la instancia** del objeto que se ha creado en el caso de los botones btnNewButton y btnNewButton_1, pero este es solo el nombre de la variable en Java; el **texto** que **se muestra al usuario** puede ser diferente y, en la mayor parte de los casos, lo será. Este se muestra **entre comillas**.
- **Propierties:** Cada uno de los componentes dispone de diferentes propiedades que podrán ser modificadas desde la ventana que se encuentra a la izquierda de la paleta de elementos (Palette).

  Una de las propiedades más importantes es la llamada **Variable**, que recoge el **nombre de la instancia** del componente creado en código Java. Por ejemplo, si se inserta un componente de tipo botón (JButton), el nombre que recibe la nueva instancia del objeto sería btnNewButton. Si se incluye otro botón, este nombre será diferente: no pueden existir dos elementos con el mismo nombre.

  En el caso de los botones, otras de las propiedades de aspecto que aparecen y que permiten definir la apariencia son: ‘background’, para seleccionar el color de fondo del botón; ‘enabled,’ que permite habilitar o deshabilitar su funcionalidad; ‘font’ y ‘foreground’, que definen el tipo de letra, el tamaño y el color, entre otras.

Creación de un JFrame

En el siguiente código se muestran cada uno de los pasos descritos en el planteamiento mediante el uso de los métodos setSize, setVisible y setDefaultCloseOperation, y se definen todos los parámetros de diseño necesarios para generar una ventana desde cero.

```
import javax.swing.*;

public class MiPrimeraVentana {

	public static void main(String[] args) {

	JFrame f = new JFrame("Mi primera ventana");

	f.setSize(400, 400);
	f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	f.setVisible(true);
	}
}
```

El resultado del código anterior es una ventana como la que se muestra en la siguiente imagen, en la que se puede apreciar las dimensiones establecidas de 400 x 400 píxeles de ancho por y alto y el nombre de la ventana.
Es importante destacar que unas de las principales diferencias a la hora de crear un JFrame directamente con el código es que no estará habilitado el modo Design.

Creación de un botón

Para crear un JFrame seleccionamos en el menú File -> New -> Other -> JFrame. Como podemos ver, esta clase crea el siguiente código implementado en la vista 'Source'. Además, por defecto, se importa la librería AWT:

Añadir objetos de tipo JButton: Desde la vista Design añadimos un componente JButton y modificamos su propiedad 'text' con el contenido «Aceptar». Repetimos el proceso para el botón «Cancelar» y lo colocamos en otro de los 'containers'.

¿Qué es GitHub?

Es un repositorio de desarrollo software que utiliza un sistema de control de versiones. Es muy útil en proyectos que tienen una gran calidad de archivos de código.

¿Qué ventajas tiene usar GitHub?

Permite el **trabajo colaborativo** entre desarrolladores, comparar versiones de un proyecto, crear copias de seguridad, mostrar gráficas sobre el flujo de trabajo de cada programador e incluso como si fuera una red social, realizar suscripciones y tener seguidores.

En la actualidad se ha convertido en una herramienta muy importante tanto desde el punto de vista educativo como profesional, dado que permite trabajar en equipo sin estar de manera presencial en un mismo sitio.

Clases y componentes

La distribución de este tipo elementos depende de los llamados **Layout. Definen** la **ubicación exacta de los elementos**.

Una aplicación puede presentar más de una ventana.

Encontramos JFrame y **JDialog**.

JDialog establece los llamados **diálogos modales o no modales.**

Área de diseño

Utilizamos el entorno Eclipse:

Conocer en profundidad todas las funcionalidades del área de la vista de diseño es fundamental para un correcto desarrollo.

Se crea la **ventana** utilizando la clase contenedora **JFrame** desde el **menú File**.

Source: (**código en Java** del JFrame creado).

Design (Vista de diseño). Las partes principales de la vista de diseño son:

- **Zona de diseño**: sobre la que se sitúan los componentes de la interfaz.
- **Palette**: aquí se encuentran todos los **elementos** utilizados para la implementación de la interfaz, que son colocados sobre la ventana de la zona de diseño. Cada vez que tomamos uno de estos elementos y lo colocamos en la pestaña '**Source**', aparece su **código de programación**.
- **Components**: mapa de **navegación** que muestra un resumen de **todos los elementos insertados** en la zona de diseño.
- **Propiedades**: si se selecciona cualquier **componente** en esta ventana, se muestran todas las propiedades del elemento que **permiten definir** su apariencia, entre otras. En cambio, si no se pulsa sobre ningún elemento aparece en blanco.

REPASO LENGUAJES ORIENTADO A OBJETOS

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 utilizando 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 un par de paréntesis. Un constructor es sintácticamente muy semejante a un método. El **constructor** puede recibir **argumentos**; de esta forma, podrá crearse más de un constructor en función de los argumentos que se indiquen en su definición. Aunque el constructor no haya sido definido explícitamente, en Java siempre existe un **constructor por defecto** que presenta el nombre de la clase y no recibe ningún argumento.

Métodos

Los métodos definen el comportamiento de un objeto.

Los métodos pueden recibir argumentos o no.

Devolverán un valor o **realizarán alguna modificación** sobre los atributos de la clase.

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 (get(), set())**.

Los atributos definen las características del objeto.

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:

- Constructor,
- atributos,
- métodos.

JFrame y JPanel

JFrame se utiliza, sobre todo, para definir la **pantalla principal** de una aplicación.

Para generar las **pantallas secundarias** es común utilizar **JDialog**, prácticamente idénticas a JFrame.

Al igual que ocurre con todas las clases en Java, es necesario utilizar un **constructor** para crear una instancia del objeto, en el caso de Jframe encontramos varios, como **Jframe()** o **Jframe (String nombreVentana)**, entre otros.

Otros **métodos importantes** en Jframe son aquellos que permiten establecer las **dimensiones** exactas de la ventana, la acción de **cierre** y habilitar su **visibilidad**.

Complementando a JFrame, tenemos un panel o **lienzo** denominado **JPanel**. Este es un bloque «invisible» que se sitúa sobre una ventana. Consiste en un **contenedor de componentes** sobre el que vamos a ubicar todos los elementos necesarios, sin tener que colocarlos directamente sobre la ventana JFrame.

Para **crear un panel**, desde el menú File, New, pulsamos Other y se busca en la **carpeta WindowBuilder** el tipo JPanel.

Gracias a los paneles podemos tener más organizada la interfaz gráfica. La **distribución de estos paneles constituye un sistema de capas o Layout** que se verá más adelante.

El resto de los componentes se colocan dentro del lienzo creado.

```
// Panel
JPanel panelSecundario = new JPanel();
// dentro de un JPanel principal (panelPpal)
panelPpal.add(panelSecundario);
// etiqueta
JLabel jlabel1 = new JLabel("hola");
// dentro del JPanel secundario
panelSecundario.add(jlabel1);
```

JDialog

Si la aplicación 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 sus padres**, ya sean 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**.

```
public DialogHola() {
		setBounds(100, 100, 450, 300);
		getContentPane().setLayout(new BorderLayout());
		contentPanel.setLayout(new FlowLayout());
		contentPanel.setBorder(new EmptyBorder(5, 5, 5, 5));
		getContentPane().add(contentPanel, BorderLayout.CENTER);
		{
			JPanel buttonPane = new JPanel();
			buttonPane.setLayout(new FlowLayout(FlowLayout.RIGHT));
			getContentPane().add(buttonPane, BorderLayout.SOUTH);
			{
				JButton okButton = new JButton("OK");
				okButton.setActionCommand("OK");
				buttonPane.add(okButton);
				getRootPane().setDefaultButton(okButton);
			}
			{
				JButton cancelButton = new JButton("Cancel");
				cancelButton.setActionCommand("Cancel");
				buttonPane.add(cancelButton);
			}
		}
	}
```

En este primer diseño aparecen dos **botones**, **Ok** y **Cancel**.

Los **diálogos modales** son aquellos que **no permiten** que **otras ventanas de diálogo** se abran **hasta** que la que se encuentra abierta **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.

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);

*el método: .**setModal(boolean b)** modificará este atributo.*

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 clic 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.

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

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

En el siguiente ejemplo, al pulsar el botón “saludar” desde la ventana principal implementada con una clase JFrame, nos lleva a una segunda ventana también de tipo JDialog, la cual muestra un mensaje en una etiqueta de texto.

Clase Frame:

```
JButton btAbreDialogHola = new JButton("saludar");
btAbreDialogHola.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // en el método actionPerformed lanzamos el DialogHola
                DialogHola hola = new DialogHola();
                hola.setVisible(true);
                dispose();
            }
});
```

Clase Jdialog:

```
public DialogHola() {
		setBounds(100, 100, 450, 300);
		getContentPane().setLayout(new BorderLayout());
		setTitle("Hola");
		setResizable(false);
		contentPanel.setLayout(new FlowLayout());
		contentPanel.setBorder(new EmptyBorder(5, 5, 5, 5));
		getContentPane().add(contentPanel, BorderLayout.CENTER);
		saludo = new JLabel("¡¡¡Holaaaa qué pasaaaaa!!!");
		getContentPane().add(saludo);

		{
			JPanel buttonPane = new JPanel();
			buttonPane.setLayout(new FlowLayout(FlowLayout.RIGHT));
			getContentPane().add(buttonPane, BorderLayout.SOUTH);
			{
				JButton okButton = new JButton("OK");
				okButton.addActionListener(new ActionListener() {
					public void actionPerformed(ActionEvent e) {
					}
				});
				okButton.setActionCommand("OK");
				buttonPane.add(okButton);
				getRootPane().setDefaultButton(okButton);
			}
			{
				JButton cancelButton = new JButton("Cancel");
				cancelButton.setActionCommand("Cancel");
				buttonPane.add(cancelButton);
			}
		}
	}
```

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

Componentes

Son los elementos que se sitúan en la ventana, directamente sobre el JFrame y JDialog o sobre un JPanel. Hay que prestar especial atención a aquellas propiedades que tienen un mismo nombre, puesto que no realizan la misma acción en todos los elementos.

JButton

Permite crear un objeto de tipo botón dentro de una interfaz gráfica en Java.

Algunas propiedades:

Background	El color de fondo del botón. Se muestra solo si es opaco
Enabled	True/false determina si el botón está activo o no
Font	Fuente del tipo de letra y tamaño
Foreground	Color del texto
HorizontalAlignment verticalAlignment	Alineación vertical y horizontal del texto con respecto al botón
Text	Texto que aparece dentro del botón
Icon	Carga imagen como fondo del botón

Para utilizar este componente es necesario importar los paquetes: import **javax.swing** e import **java.awt.event**. El segundo se utiliza para que, al pulsar el botón, se **detecte la ocurrencia de un evento** y se derive una acción.

Ahora crearemos un botón y modificaremos algunas propiedades:

```
JButton btAbreDialogHola = new JButton("saludar");
panelSecundario.add(btAbreDialogHola);
// establecemos tipo y tamaño de letra
btAbreDialogHola.setFont(new Font("Monospaced", Font.BOLD, 12));
// establecemos el color de fondo del botón
btAbreDialogHola.setBackground(Color.MAGENTA);
// determinamos el color del texto
btAbreDialogHola.setForeground(Color.white);
```

JLabel

Se trata de un elemento de texto. **Puede llegar a albergar** también **imágenes**, **iconos** o **texto**. Sus propiedades características son:

Background	El **color** de la etiqueta si está **deshabilitada**
Enabled	Habilita la etiqueta
Font	Fuente del tipo de letra y tamaño
Foreground	**Color** del texto si etiqueta **habilitada**
HorizontalAlignment verticalAlignment	Alineación vertical y horizontal del texto con respecto a la caja de la etiqueta
Text	Texto que aparece dentro de la etiqueta
Icon	Permite cargar una imagen

Hay que prestar **especial atención** a los valores **background y foreground**, ambos definen el color del texto: el primero cuando está **habilitado** y el segundo cuando está **deshabilitado**.

JTextField

El elemento JTextField, se utiliza como contenedor de **una línea de texto.**

Algunas de sus propiedades:

background	Color de **fondo** de la caja de texto
foreground	Color del **texto**
columns	**Tamaño** de la caja de texto
enabled	Habilita el campo de texto
editable	Permite al usuario modificar el contenido
font	Fuente del tipo de letra y tamaño
horizontalAlignment	Alineación horizontal del texto
text	Texto que aparece al inicio en la caja

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 cuál 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**.

JRadioButton

Para indicar varias opciones, de las que **solo se podrá escoger una**, es decir, que 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 al de 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.

En el siguiente ejemplo se han creado dos elementos y se han asociado en un un ButtonGroup llamado bgSexo:

```
JRadioButton rbHombre = new JRadioButton("hombre");
rbHombre.setBounds(31, 201, 149, 23);
panelPrincipal.add(rbHombre);
JRadioButton rbMujer = new JRadioButton("mujer");
rbMujer.setBounds(31, 228, 149, 23);
ButtonGroup bgSexo = new ButtonGroup();
bgSexo.add(rbMujer);
bgSexo.add(rbHombre);
```

JComboBox

Menús desplegables: 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 escriben 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**.

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 por defecto y ocupan todo el contenedor. El uso de los layout nos permite modificar el tamaño de los componentes y su posición de forma automática.

Flow Layout

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

```
FlowLayout fl_contentPane = new FlowLayout(FlowLayout.LEFT, 10, 10);
contentPane.setLayout(fl_contentPane);
```

Grid 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 la column y row, que definen el número exacto de columnas y filas.

Para la creación de este sistema de rejilla, se utiliza un **constructor** que necesita recibir el valor exacto de **filas** y **columnas** que tendría la interfaz, **GridLayout(int numFilas, int numCol)**.

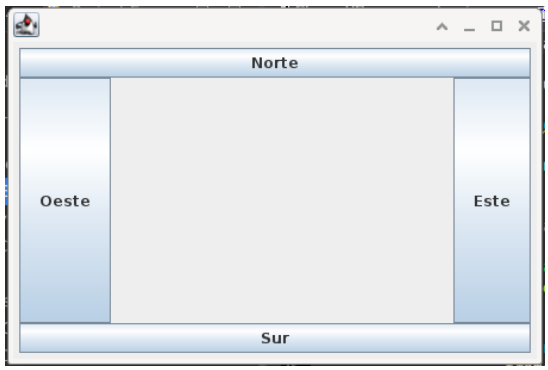
Cualquiera de los elementos 'Layout' presentan como propiedad común el valor de **vgap y hgap**, que definen la **distancia entre elementos** que se crea tanto en vertical como en horizontal.

```
contentPane.setLayout(new GridLayout(2,2));

		lblNewLabel = new JLabel("New label");
		contentPane.add(lblNewLabel);

		lblNewLabel_1 = new JLabel("New label");
		contentPane.add(lblNewLabel_1);

		lblNewLabel_2 = new JLabel("New label");
```

```
            contentPane.add(lblNewLabel_2);

            lblNewLabel_3 = new JLabel("New label");
            contentPane.add(lblNewLabel_3);
```

Border Layout

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**.



Ahora bien, **desde el código** se sitúan atendiendo a su situación (**NORTH**, **SOUTH**, **EAST**, **WEST**, **CENTER**).

```
setContentPane(contentPane);
contentPane.setLayout(new BorderLayout(0, 0));
JButton btnNewButton = new JButton("Norte");
contentPane.add(btnNewButton, BorderLayout.NORTH);
JButton btnNewButton_1 = new JButton("Oeste");
contentPane.add(btnNewButton_1, BorderLayout.WEST);
JButton btnNewButton_2 = new JButton("Este");
contentPane.add(btnNewButton_2, BorderLayout.EAST);
JButton btnNewButton_3 = new JButton("Sur");
contentPane.add(btnNewButton_3, BorderLayout.SOUTH);
```

GridBag Layout

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

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

DESARROLLO DE INTERFACES

TÉCNICO EN DESARROLLO DE APLICACIONESMULTIPLATAFORMA

Interfaces a partir de XML

El lenguaje XML se suele usar para:

- **Intercambiar de información entre aplicaciones**.
- **Computación distribuida**: intercambiar información entre diferentes ordenadores a través de **redes**.
- **Información empresarial**: generar **interfaces empresariales estructurando** los **datos** de la forma **más apropiada** para cada empresa.

Conseguiremos **generar interfaces** a partir de documentos XML.

Lenguajes de descripción de interfaces basados en XML

XML (**eXtensible Markup Language**) maneja datos:

- **estructurándolos.**
- **almacenándolos.**
- **intercambiándolos.**

Es un metalenguaje, y **define** otros **lenguajes:**

XHTML

eXtensible HyperText Markup Language: similar a HTML, pero **más robusto y aconsejable** para la modelación de **páginas web**.

Elementos obligatorios:

- <! DOCTYPE>
- <html **xmlns**> atributo **xmlns.**
- <html>, <head>, <title> y <body>.

Características de diseño de sus elementos:

- Correctamente **anidados**.
- **Cerrados**.
- En **minúsculas** los elementos y los nombres de los atributos.
- Los **valores** de los **atributos siempre** se deben citar.
- **Prohibido minimizar atributos**.

En el siguiente ejemplo se muestra el prototipo de un documento redactado con formato XHTML en el que podemos comprobar que se cumplen todas las características descritas.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
        <cabeza>
                <title>Un documento XHTML</title>
        </cabeza>
        <cuerpo>
                <h1>Cabecera principal del documento</h1>
                <p>Nuestro primer párrafo</p>
                <h2>Cabecera secundaria</h2>
                <p>Otro párrafo con contenido distinto</p>
        </cuerpo>
</html>
```

GML

GML (**Geography Markup Language**). Puede **recibir un formato** que define el **tipo de texto** que es (h1, p, lo, li...). Estos tipos de documentos están compuestos de **marcas precedidas de doble punto(:)**. Ejemplo:

:h1.Lenguaje XML

:p.Lenguajes basados en XML

:ol

:li.GML

:li.MathML

:li.RSS

:li.SVG

:li.XHTML

:eol.

MathML

El lenguaje MathML (**Mathematical Markup Language**) se usa junto con el lenguaje XHTML y se basa en el intercambio de información de tipo matemático entre programas.

```
<math>
      <mi>x</mi> <mo>=</mo>
      <mrow>
            <mfrac>
                  <mrow>
                        <mo>−</mo>
                        <mi>b</mi>
                        <mo>±</mo>
                        <msqrt>
                              <msup><mi>b</mi><mn>2</mn></msup>
                              <mo>−</mo>
                              <mn>4</mn><mi>a</mi><mi>c</mi>
                        </msqrt>
                  </mrow>
                  <mrow> <mn>2</mn><mi>a</mi> </mrow>
            </mfrac>
      </mrow>
      <mtext>.</mtext>
</math>
```

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

RSS

Really Simple Syndication: Difusión de información entre los usuarios suscritos a una **fuente de contenidos** actualizada frecuentemente.

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="1.0">
        <channel>
                <title>Última hora</title>
                <description>
                        Noticia importante
                </description>
                <link>
                        https://elpais.com/ultimas-noticias/
                </link>
                <lastBuildDate>
                        Mon, 06 Jan 2020 00:10:00
                </lastBuildDate>
                <pubDate>
                        Mon, 06, Jan 2020 16:20:00 +0000
                </pubDate>
        </channel>
</rss>
```

XSLT

EXtensible Stylesheet Languaje for Transformation: Parecidas a **hojas de estilo** CSS, pero **dirigidas a XML**. Presenta un funcionamiento más completo que CSS, puesto que **permite agregar o eliminar elementos y atributos** desde un archivo.

Además, permite **realizar pruebas** e, incluso, tomar **decisiones** sobre los elementos que se han de **mostrar u ocultar**.

SVG

Scalable Vector Graphics: Representa elementos geométricos vectoriales, imágenes de mapa de bits y texto.

circle

```
<!DOCTYPE html>
<html>
<body>
        <svg height=”100” width=”100”>
        <circle cx=”50” cy=”50” r=”40” stroke=”black”
        stroke-width=”3” fill=”cyan” />
        </svg>
</body>
</html>
```

rect

```
<svg width=”60” height=”60”>
        <rect x=”0” y=”0” width=”60” height=”60” fill=”red”/>
</svg>
```

ellipse

```
<svg width=”60” height=”60”>
        <ellipse cx=”30” cy=”30” rx=”20” ry=”16” fill=”orange”/>
</svg>
```

polygon

```
<svg width="60" height="60">
        <polygon fill="green" stroke="black" stroke-width="2" points="05,30
        15,10,25,30"/>
</svg>
```

El documento XML.
Intro

La **primera línea del fichero debe ser la siguiente**:

<?xml version="1.0" encoding="UTF-8"?>

• Version: Versión de XML.

• Encoding: **Juego de caracteres**.

El resto del documento XML se escribirá con **etiquetas**, y siempre hay que abrirlas y cerrarlas:

<etiqueta> (...) </etiqueta>

El conjunto formado por las etiquetas (apertura y finalización) y el contenido se conoce como **elemento** o **nodo** (en el caso de hacer una representación jerárquica de los datos).

Por ejemplo, el conjunto

<nombre> Lucas </nombre>

es un elemento o nodo, con la etiqueta 'nombre' y el contenido 'Lucas'.

Estructura de un **elemento o nodo** (pueden estar anidados con otros elementos en su interior):

<nombre-del-elemento *atributo* = "valor-del-atributo" >Contenido del elemento</nombre-del-elemento>

Estructura anidada de manera **jerárquica**. **Etiquetas** para que **queden todas cerradas**. Las etiquetas de los descendientes/hijos deben cerrarse antes que la del padre.

Las **etiquetas** abiertas deben haber sido **cerradas en el orden adecuado**. Los valores de los datos o **contenidos** de los nodos se encuentran **entre** el texto que indica la **apertura** de la etiqueta **y** lo que indica el **cierre**.

Etiquetas

Son marcas que sirven para diferenciar un contenido específico del resto del contenido del documento.

Empieza con ‘<”, continúa un nombre identificativo, y termina con “>”.

Siempre debe empezar por una letra, pero, a continuación, pueden utilizarse: **letras, dígitos, guiones, rayas, punto o dos puntos**. Existen **tres tipos de etiquetas**:

- **Start-Tag**: Etiquetas de apertura. (<etiqueta>)
- **End-Tag**: Etiquetas de cierre, similar a las de apertura, pero **comienzan por “/”**. (</etiqueta>)
- **Empty-Tag**: Etiquetas vacías, que terminan por “/”. (<etiqueta_vacía />)

Atributos

Es un componente de las etiquetas. Consiste en un **par ‘nombre=valor’**. Se puede encontrar en las etiquetas de apertura o en las etiquetas vacías, pero no en las de cierre. **No pueden existir dos atributos con el mismo nombre**, y todos los atributos de un elemento **siempre son únicos**. Por ejemplo:

<conductor nombre=”Marcos” apellido-1=”García” apellido-2=”López” />

En este caso tenemos tres atributos únicos, nombre, apellido-1 y apellido-2.

En cambio, en el siguiente caso, **no sería correcto**, dado que tenemos el atributo apellido repetido:

<programador nombre=”Marcos” apellido=”López” apellido=”García”>

Valores

El **atributo** de un elemento XML **proporciona información** acerca del elemento, es decir, sirve para definir las propiedades de los elementos. La **estructura de un atributo** XML es siempre un **par de ‘nombre=valor’**.

```
<biblioteca>
    <texto tipo_texto="libro" titulo="Soft Skills:The software developer's life manual" editorial="Manning Publications">
        <tipo>
            <libro isbn="978-1617292392" edicion="1" paginas="503"/>
        </tipo>
        <autor nombre="John Sonmez"/>
    </texto>
</biblioteca>
```

En el ejemplo observamos que los ***elementos*** aparecen coloreados en rojo (biblioteca, texto, tipo), los **nombres** de los atributos en blanco (tipo_texto, título, editorial, isbn, edición, páginas) y sus ***valores*** en azul.

Eventos

Gracias a los eventos el usuario podrá llevar a cabo una determinada funcionalidad. Eventos dan lugar a **interfaces dinámicas**.

Trata las diferentes formas de interacción entre el usuario y la aplicación. De todos los posibles eventos en una aplicación, elegiremos cuales queremos usar para una determinada funcionalidad.

Ejemplos de eventos son:

- **MouseMove**: se produce al mover el ratón por encima de un control.
- **MouseDown**: se produce al pulsar **cualquier botón** del ratón.
- **Change**: se produce al **cambiar el contenido del control**.
- **Click**: se produce al hacer clic sobre el control con el **botón izquierdo** del ratón.
- **GetFocus**: se produce cuando el elemento **recibe el foco** de atención, normalmente se utiliza para introducir datos o realizar alguna operación en tiempo de ejecución.
- **LostFocus**: se produce cuando el elemento de control **pierde** el foco.

Un ejemplo de uso de eventos sería si queremos que un texto se ponga de color rojo al situarnos encima, y de color gris al salir, podríamos usar **MouseEntered** y **MouseExited**; el primer evento se encargaría de poner el texto de color rojo y, el segundo, de ponerlo de color gris.

Herramientas para crear interfaces de usuario multiplataforma

Notepad ++

Reconoce la sintaxis de **múltiples lenguajes** de programación. Es gratuito y disponible para **Linux y Windows**.

Ha triunfado bastante entre la comunidad de desarrolladores web por las características que ofrece y por lo ligero que es (ocupa poco espacio y es muy rápido). Se puede extender a través de plugins. Posee uno llamado XML Tools que añade un nuevo menú con numerosas opciones como, por ejemplo, validar un documento XML con su DTD. Su interfaz es minimalista, pero los desarrolladores pueden personalizarla.

Atom

Herramienta multinivel, tanto para novatos como para **profesionales**. Actualmente es uno de los editores **preferidos** para programadores. Se pueden añadir lenguajes que no se incluyen de serie o añadir distintos tipos de interfaces gráficas.

Cada ventana de Atom es, en esencia, una **página web renderizada localmente**, y el espacio de trabajo se compone de paneles que pueden recolocarse de manera flexible para que la programación resulte más cómoda.

Teletype (Atom)

Para colaborar en tiempo real , permite que muchos desarrolladores puedan editar un archivo a la vez, en tiempo real, de forma colaborativa.

Su funcionamiento se basa en que el **usuario con rol de anfitrión** comparte su código con el resto de usuarios. Para lograrlo, se utiliza un código generado por el anfitrión y que este comparte con los invitados. Cada uno de ellos podrá editar el código compartido en tiempo real y visualizar las modificaciones del resto del equipo.

Git y GitHub (plugin en Atom)

Permiten controlar distintas versiones de un proyecto mientras se está desarrollando. El proyecto en el que estamos trabajando podrá **sincronizarse automáticamente** con el repositorio Git y podremos visualizar en todo momento **si estamos trabajando en la misma versión** que se encuentra en el **repositorio** o qué **diferencias** existen.

Adobe Dreamweaver CC

Admite tanto el **método textual** como el **WYSIWYG (What You See Is What You Get)**. Esta es una frase aplicada a los editores de código que permiten escribir un documento **mostrando directamente el resultado final**. ¿Presentación visual en vivo? ¿o seguir el camino clásico?

Admite todos los lenguajes de programación importantes. Está disponible para **Windows y OS X**. Permite **confirmar el código y accesibilidad** de la página, característica que puede facilitarles seguir las **pautas** de **accesibilidad** de contenido web.

Visual Studio Code

De los **más utilizados**. Trabaja con **varios lenguajes** de programación y está disponible para **Windows, Linux y macOS**.

Una de sus principales características es el resaltado de sintaxis, además de la finalización inteligente de código, la interfaz personalizable y que es gratuito (aunque existe la versión de pago). Pertenece al software de Visual Studio y una de las novedades más potentes que ofrece es el servicio **Live Share**, extensión que permite compartir código base con un compañero de equipo, de forma que se puede **colaborar en tiempo real** (parecido al *teletype* de atom).

Generación de código para diferentes plataformas

El intercambio de datos con XML ayuda a que los datos pueden ser leídos por **diferentes plataformas**.

Al ser XML un lenguaje independiente de la plataforma, significa que **cualquier programa diseñado para lenguaje XML** puede leer y **procesar los datos XML** independientemente del hardware o del sistema operativo que se esté utilizando.

Es por eso una de las tecnologías más utilizadas como base para el **almacenaje** de contenidos, como modelo de representación de **metadatos**, y como medio de **intercambio** de contenidos:

• **XML para el almacenamiento de contenidos**: en los últimos años está creciendo la demanda de **bases de datos XML nativas**, es decir, bases de datos que almacenan y gestionan documentos XML directamente, sin ningún tipo de transformación previa.

• **XML como medio de intercambio de contenidos**: la integración que permite el lenguaje XML en diferentes plataformas se basa en la facilidad de intercambio de contenidos dado que, al utilizar documentos basados en este lenguaje, se puede procesar para múltiples fines: como integración en una base de datos, visualización como parte de un sitio web o mensajes entre aplicaciones.

• **XML para la representación de metadatos**: lo más importante para representar metadatos es el sistema de **indexación y recuperación**, para poder discriminar dentro de un contenido los **elementos o atributos** que se desea recuperar.

La generación de código para diferentes plataformas es posible *debido a que:*

- *Es un estándar* ***abierto***, ***flexible*** *y utilizado a* ***nivel global*** *para almacenar, publicar e intercambiar todo tipo de información.*
- *Proporciona* ***portabilidad*** *y* ***facilita la gestión de la información*** *a través de distintas plataformas.*
- *Permite que* ***diversas aplicaciones de datos*** *puedan funcionar de forma* ***independiente****.*
- *Es soportado por multitud de aplicaciones en diferentes plataformas. Además, existen varias* ***bibliotecas para diferentes lenguajes*** *de programación que permiten desarrollar nuevas aplicaciones. Es un* ***metalenguaje*** *de "carácter* ***universal****".*
- *El* ***formato*** *es* ***legible*** *tanto por humanos como por sistemas informáticos.*

XML permite el ***intercambio*** *entre plataformas del conjunto de* ***datos estructurados*** *que componen una interfaz de* ***forma simple, rápida y segura****.*

DESARROLLO DE INTERFACES

TÉCNICO EN DESARROLLO DE APLICACIONESMULTIPLATAFORMA

Creación de componentes

Un componente es un elemento **JavaBean**, con características que definen su funcionamiento (introspección, reflexión, customización y comunicación activa entre el usuario y la interfaz a través de los eventos).

Para agregar un componente se genera un archivo .jar que puede ser importado en cualquier entorno de desarrollo, permitiendo así el uso de sus componentes.

Librería Graphics: realiza múltiples diseños gráficos sobre ventanas y paneles.

Es posible crear **nuevos componentes** a partir de uno ya existente: modificando sus características y propiedades, adaptando sus elementos.

Componentes
Reutilización de software

Crear un **componente** consiste en crear un **bloque de software reutilizable**, es decir, se va a implementar un **elemento** a través de un determinado lenguaje de programación que va a poder ser **utilizado en cualquier otro proyecto**.

Cuando hablamos de reutilización del código, nos referimos a la **reutilización de librerías**, **frameworks** o kit de **herramientas**.

Utilizando como base **elementos ya desarrollados** se podrán crear otros nuevos que presenten otras características que el componente tomado como referencia no incorpora. Se toman **como base los ya implementados** y se les incorporan ciertas **mejoras**.

Beneficios de **reutilizar módulos** software:

- **Se reducen los costes** del proyecto (coste = tiempo de desarrollo).
- Las **pruebas** de software se **simplifican**, solo será necesario someter a pruebas a los **nuevos desarrollos**.
- Mejora la **calidad** del software. Cada nueva implementación, se agregan nuevas funcionalidades más específicas.

Reutilización de software, **tres niveles**:

1- a nivel de **clase** (clases y algoritmos).

2- a nivel de **diseño** (patrones de diseño).

3- a nivel de **arquitectura**.

Características de los componentes

Definición: Módulo de código ya **implementado y reutilizable** que puede interactuar con otros componentes software a través de las interfaces de comunicación.

De componentes más generales se podrán **implementar** múltiples **versiones modificadas** (comenzar con componentes software generales...).

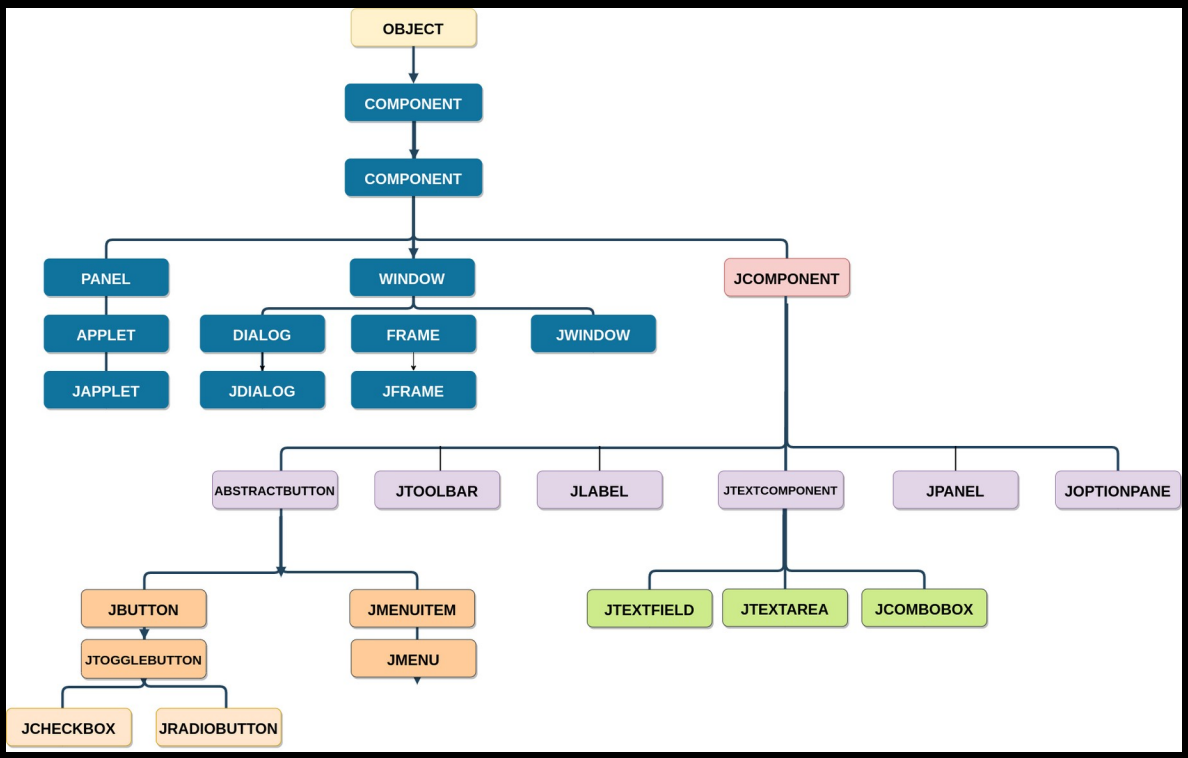
En diseño gráfico con **JSwing**, un **componente** es alguno de los **elementos** que se sitúan en la ventana, directamente sobre el **JFrame** y **JDialog**, o sobre un **JPanel**, y que le aporta funcionalidad a la interfaz.

Para diseñar un **buen componente**, es deseable que presente ciertas **características**:

- La **implementación** puede estar realizada con **cualquier lenguaje** de programación, pero ha de estar **completa**.
- Constituye un **módulo reutilizable**, ya **compilado** (archivo .jar en java).
- Distribución: **paquete ejecutable**.

POC (Programación Basada en Componentes): metodología de programación basada en el uso de **elementos reutilizables**.

Componentes visuales (librería **Jswing)** (reutilizables y personalizables):



Propiedades / atributos

Las **propiedades de un componente** definen los **datos públicos** que forman la **apariencia** y **comportamiento** del objeto. Las propiedades pueden **modificar** su valor a través de los **métodos** que definen el comportamiento de un componente.

Los **métodos clave** que permiten **analizar el contenido de una propiedad o atributo** son los de tipo get, mientras que para modificar su valor se utilizan los métodos set.

- **Ámbito público**: propiedad utilizada desde cualquier parte de la **aplicación**.
- **Ámbito privado**: propiedad solo es accesible desde la clase donde se ha creado.
- **Ámbito estático**: propiedad utilizada **sin** la necesidad de crear una **instancia del objeto** al que está referida.

Se distinguen principalmente dos **tipos de propiedades**:

- Las **propiedades simples** (un solo v**alor**):

```
JCheckBox checkBox = new JCheckBox(“OK”);
checkBox.setSelected(true);
```

- Las **propiedades indexadas** (**conjunto de valores** en forma de **array)**:

```
JComboBox comboBox = new JComboBox();
comboBox.setModel(new DefaultComboBoxModel(new String[] {“1”, “2”, “3”}));
```

Los **atributos** se utilizan para **almacenar los datos internos** y de **uso privado** de una **clase** u objeto.

JavaBean

Son componentes software que permiten su **reutilización en las interfaces de usuario java**. Pueden estar representados gráficamente o no. **Componentes y elementos de la paleta** gráfica (entorno de diseño) son **módulos de tipo JavaBean**.

Es el primer modelo para el desarrollo de componentes de Java e implementa el modelo Propiedad-Evento-Método, sus **características distintivas** son:

- los métodos,
- las propiedades
- y los eventos.

Características comunes:

• **Introspection.** El IDE podrá **analizar** en profundidad el **funcionamiento** concreto del JavaBean. La clase **BeanInfo** ofrece un soporte en el que se **recoge información** sobre características adicionales.

• **Persistence.** Podrá ser almacenado para ser utilizado posteriormente en **cualquier proyecto**. Se logra gracias a la **serialización** del componente (**implements Serializable**).

• **Customization.** **Propiedades** y **comportamiento** modificable en su implementación.

• **Events**. Podrá tener **asociadas** unas **acciones** como respuesta a un estímulo. Es posible la **comunicación** entre **componentes desarrollados** de forma completamente **independiente**.

Diferencia entre un JavaBean y las clases de Java

Un JavaBean es una clase de Java que **agrupa en un objeto a varios objetos** como si fuese una especie de cápsula. Tiene que cumplir las siguientes **reglas**:

- Debe tener un **constructor público sin argumentos**.
- **Ninguna variable** de instancia **pública**.
- Las propiedades deben ser accesibles con los **métodos get y set**.

Los JavaBeans se crearon en un primer momento **para las herramientas gráficas** que se utilizan en el desarrollo de aplicaciones, pero esa tendencia ha cambiado **últimamente** ya que ahora se usan en muchos tipos de programas para **encapsular código** y que sea **reutilizable**.
Un Bean puede tener **clases asociadas**, por ejemplo, una clase BeanInfo proporciona información sobre el Bean, sus propiedades y sus eventos.
Un JavaBean es una clase Java, pero **una clase Java no tiene por qué ser un JavaBean**.

Creación de un nuevo componente JavaBean

Utilizar otros **componentes ya desarrollados** e introducirles **mejoras** y otros **cambios** que se adapten a nuevas casuísticas.

Es necesario crear un **nuevo proyecto** para la creación de un **nuevo componente**. El nuevo componente extenderá del base:

```
public class NuevoComponente extends ComponenteBase{};
```

Para la creación de un nuevo componente, la clase que lo implementa ha de ser de **tipo JavaBean**, debe cumplir **dos características** básicas:

1. Tiene que implementar **Serializable**:

```
public class NuevoComponente extends ComponenteBase implements Serializable {}
```

2. Ha de tener un **constructor sin parámetros**:

```
public class NuevoComponente extends ComponenteBase implements Serializable {
    public NuevoComponente(){}
}
```

Cada nuevo comportamiento implementa una nueva funcionalidad (**métodos**).

Redefinir el comportamiento: utilizar la palabra reservada **@Override** seguido del mismo **nombre del método** en la **clase de referencia** (“nuevoMetodo” en el ejemplo):

```
@Override
protected void nuevoMetodo ( … )
{super.nuevoMetodo( … );
}
```

Para mantener la misma funcionalidad que el método del componente base, se utiliza el **constructor** de la clase padre de la que se hereda (**super**) y, a continuación, se añade la **nueva implementación**.

A los nuevos componentes **solo habrá que añadir las propiedades nuevas** porque ya parten con las del componente base. Se **implementan los atributos nuevos**, en la **parte superior** de la clase.

```
public class NuevoComponente extends ComponenteBase implements Serializable {
        private tipo(int, String,...) nombrePropiedad;
        public NuevoComponente(){...}
}
```

Se deberán incorporar los **métodos set y get** a la clase, y así poder establecer y obtener el valor de los atributos.

```
public class NuevoComponente extends ComponenteBase implements Serializable {
        private tipo(int, String,...) nombrePropiedad;
        public NuevoComponente(){}
        public void setNombrePropiedad(tipo nombrePropiedad){
        this.nombrePropiedad=nombrePropiedad;
        }
        public [tipo] getNombrePropiedad(){
        return nombrePropiedad;
        }
}
```

En el **menú de propiedades** de **vista de diseño** ya aparecerán los **atributos** del nuevo componente.

Integración de un nuevo componente.
Empaquetado

Para agregar un nuevo componente **a la paleta de diseño** es necesario disponer del **Jar** del componente base.

Los pasos para generar un Jar desde Eclipse son:

1. Crea un **paquete** y coloca la **clase o clases implementadas** del nuevo componente dentro del paquete.

2. Desde el menú de **herramientas**, limpiamos y **construimos el proyecto**.

    Project>Clean

    Project>Build project

3. Accedemos a la opción **Export** del **proyecto** donde se ubica el nuevo componente (pulsando con el botón derecho sobre el nombre del proyecto) y escogemos la opción **Jar File**, situada dentro de la carpeta Java. Finalmente, indicamos un nuevo **nombre para el archivo Jar** y se **guarda**.

Ahora bien, antes de poderlo utilizar en otros proyectos debemos de **modificar el archivo manifest** generado en el paquete del componente de la siguiente manera: descomprimimos el paquete .jar; dentro, en el directorio META-INF encontramos el manifest.mf, que contiene solamente: *Manifest-Version: 1.0*. Debemos dejar una línea en blanco debajo de esta única línea que ya contenía, y a continuación en la siguiente línea escribir:

Name: [y seguido, sin estos corchetes, la ruta completa de nuestra clase bean dentro del componente nuevo]

por ejemplo, si la clase en la que creamos nuestro componente se llama *NuestroComponenteNuevo.class*, sería:

Manifest-Version: 1.0.

Name: beans/NuestroComponenteNuevo.class

Java-Beans: true

Dejando la última línea en blanco, después de *Java-Beans: true.*

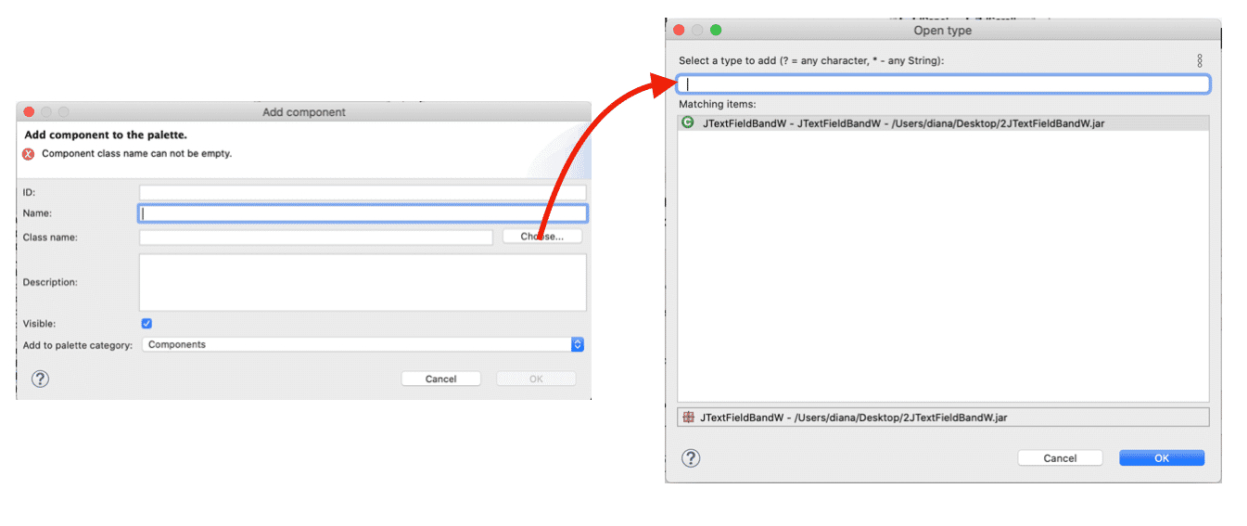
Cambiando este archivo manifiesto con el nuevo archivo ya modificado, ya tenemos el nuevo componente bean .jar listom para usar en nuestros proyectos

Para incluir el componente en un nuevo proyecto, se **añade a las librerías el archivo** Jar creado o descargado. Esto se realiza a través de la opción **Build Path** (botón derecho sobre el nuevo proyecto) y, a continuación, **Configure Build Path**. Finalmente, sobre **Classpath**, añadimos el **Jar** utilizando la función **Add external jars**.

Y se **incluye el componente en la paleta** de la zona de diseño, Palette y el nuevo elemento quedará incluido de forma permanente:

> Con el botón derecho, pulsamos sobre la **categoría** donde se va a incluir el nuevo componente y, a continuación, **Add component**.

Ahora, al **buscar el nombre de la clase**, este debe **aparecer**. Se selecciona y se pulsa OK en las dos ventanas.

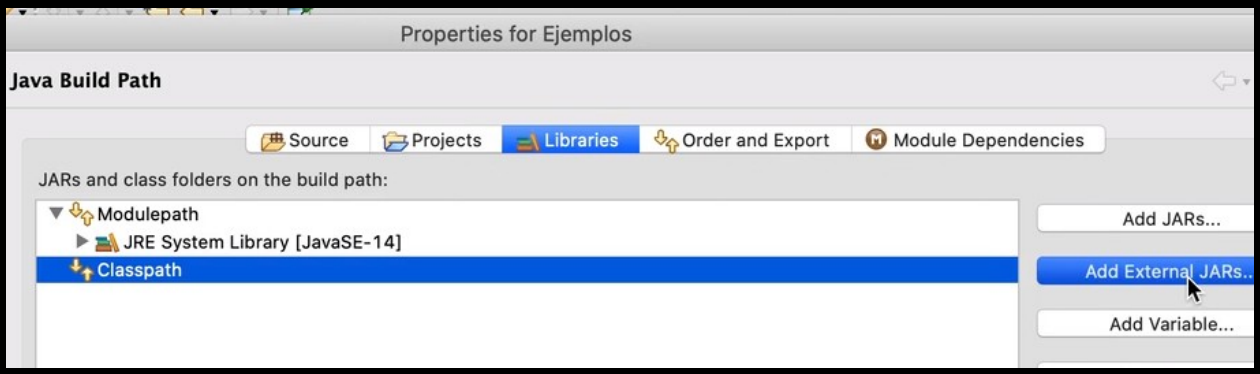


Ejemplo:

Añadir un calendario con Jcalendar, de https://toedter.com/jcalendar/:

Leemos su API... y descargamos su paquete. En el directorio lib (librerias) buscamos los archivos .jar que son los que nos interesan, y todos esos, sean los que sean, son los que importaremos a nuestro proyecto.

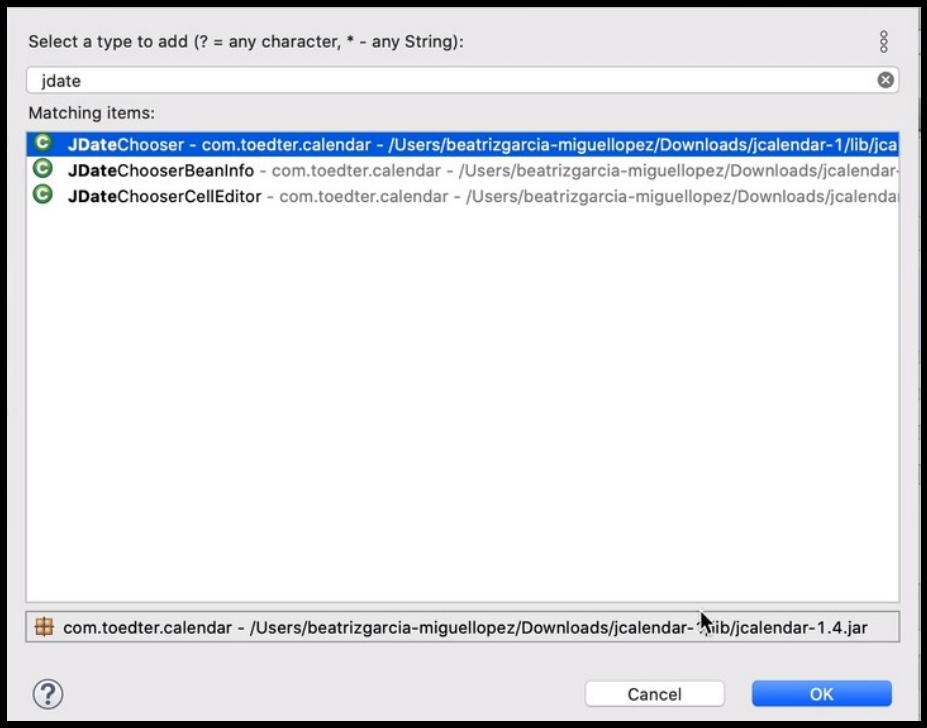
Y en nuestro proyecto, botón derecho de ratón y en el menú contextual: ***build path***, después ***configure build path***, y en la ventana que se abre, ***Add External JARs***:



Seleccionamos los archivos .jar que tenemos guardados.

Luego ya podremos importar el paquete JDateChooser, que se incluye en las librerías que acabamos de agregar al proyecto.

Para añadirlo como componente a la paleta, haciendo click derecho sobre componentes, añadir componente, buscamos la clase: JdateChooser. Y ya nos aparecerá en nuestra paleta.



Herramientas de desarrollo de componentes visuales

GIMP

Programa de edición de imágenes gratuito. Edición de representaciones digitales en forma de mapa de bits, dispone de tijeras inteligentes, filtros, la posibilidad del uso de **capas**, entre otras **muchas características**.

Microsoft Paint o Pinta

Permiten una edición de calidad para aquellos casos en los que **no** se necesita algo **demasiado profesional**. Funcionalidades para el diseño 3D.

Photoshop

Algunas características:

- Elevada **potencia de procesamiento** de imágenes.
- Elaborar diseños desde cero.
- Sistema de **capas** (multitud de efectos y tratamientos).
- Muchos tipos de **formatos**.
- Filtros, efectos, eliminación del ruido, retoque de la imagen…

Graphics y figuras

Permite dibujar sobre la interfaz manejando los píxeles como si de un lienzo en blanco se tratara. Graphics se encuentra bajo la **librería AWT**. Dos de los **métodos esenciales** para dibujar son:

paint(Graphics nombre): Se utiliza para **dibujar** sobre la interfaz la **primera vez** que se muestra, cuando se **maximiza**, o cuando **vuelve a estar visible**. Todos los componentes utilizan este método para **dibujar** su "forma". Si se crea un nuevo **componente** a partir de otro y se quiere **modificar su diseño gráfico**, se debe sobrescribir este método.

update(Graphics nombre): Actualiza los gráficos dibujados sobre la interfaz.

El **atributo Color** a través del método **setColor** (Color c), permite **definir el trazo** de la figura que se va a diseñar.

Para añadir figuras geométricas sobre la interfaz de diseño, tenemos los métodos:

drawLine (int x1, int y1, int x2, int y2): Dibuja una línea desde la posición inicial marcada por las coordenadas x1 e y1 hasta la final x2 e y2.

drawRec (int x, int y, int width, int height): Dibuja un rectángulo desde la posición inicial marcada por las coordenadas x e y, y con la altura y ancho indicados.

fillRec (int x, int y, int width, int height): Crea y rellena el rectángulo del mismo color que el definido para la línea del borde.

drawOval (int x, int y, int width, int height): Dibuja una **elipse** desde la posición inicial marcada por las coordenadas x e y, y con la **altura y ancho** indicados.

fillOval (int x, int y, int width, int height): Crea y rellena la elipse del mismo color que el definido para la línea del borde.

drawPolygon (int [] x1, int [] y1, int nPoints): Dibuja un polígono utilizando el array de coordenadas x e y pasado por parámetro.

fillPolygon (int [] x1, int [] y1, int nPoints): Dibuja un polígono relleno del mismo color que la línea de perímetro.

Ejemplo de dibujo de formas básicas con Graphics:

Este es el código base de un JFrame con su constructor:

```
import java.awt.EventQueue;
public class DibujoFormasGrafics extends JFrame {

	private static final long serialVersionUID = 1L;
	private JPanel contentPane;

	/**
	 * Launch the application.
	 */
	public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				try {
					DibujoFormasGrafics frame = new
							DibujoFormasGrafics();
					frame.setVisible(true);
				} catch (Exception e) {
					e.printStackTrace();
				}
			}
		});
	}
	/**
	 * Create the frame.
	 */
	public DibujoFormasGrafics() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setBounds(100, 100, 450, 300);
		contentPane = new JPanel();
		contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

		setContentPane(contentPane);
		contentPane.setLayout(null);

		setBounds(0, 0, 800, 600);
	}
}
```

Y sobreescribimos el **método paint**:

```
	@Override
		public void paint(Graphics graphics) {
			// Llamar al método paint de la clase superior.
			super.paint(graphics);
			// Activar el método del color que queramos, en este caso es verde
			graphics.setColor(Color.green);
			/**
			  Dibujar un óvalo desde la coordenada 'x' en 150 píxeles
			  e 'y'
			   en 70 píxeles.
			  Con un radio en el eje 'x' de 50
			 */
			graphics.fillOval(150, 70, 50, 70);
			graphics.setColor(Color.yellow);
			/**
			   Dibujar un rectángulo que comience en el eje 'x' en el píxel
			   350 y en el eje
			   'y' en el 70. De ancho que tenga 50 píxeles y de alto 70.
			 */
			graphics.fillRect(350, 70, 50, 70);
			graphics.setColor(Color.orange);
			/**
			   Dibujar un triángulo utilizando el método fillPolygon indicando
			   las posiciones en el eje 'x' de los tres vértices, las
			   posiciones de los tres vértices en el eje 'y'.
			 */
			int[] vx1 = { 600, 650, 550 };
			int[] vy1 = { 70, 120, 120 };
			graphics.fillPolygon(vx1, vy1, 3);
		}
```

Imágenes

Los componentes destinados a la creación de ventanas, paneles u otros contenedores no incluyen la opción de cargar a través de una URL una imagen de fondo. Ahora bien, utilizando los métodos propios de la clase Graphics y añadiendo ciertas modificaciones sería posible incorporar esta funcionalidad. Hablamos del **método paint**.

En el siguiente fragmento de código, en primer lugar, se realiza un llamamiento al método paint utilizando la palabra reservada super, por lo tanto, se está haciendo un llamamiento al constructor de la superclase.

```
super.paint(g);
```

A continuación, se crea una **instancia** de la clase **Toolkit**, esto permite **asociarle una imagen** a través de su **URL** completa a un **objeto** de tipo **Image**.

```
Toolkit t = Toolkit.getDefaultToolkit ();
Image imagen = t.getImage (“rutaCompleta”);
```

Para **mostrar la imagen** cargada se utiliza el método **drawImage** de la clase Graphics, el cual recibe, también por **parámetro**, la **posición** exacta en la que la imagen se va a dibujar.

```
g.drawImage(imagen, coordenadaX, coordenada Y, ancho, altura, this);
```

Si no se indica el **valor** correspondiente a las **dimensiones de la imagen**, se **cargará la imagen** con el **tamaño** con la que ha sido **guardada**. Es aconsejable **ajustarlo a la ventana**, puesto que de lo contrario la imagen se vería incompleta.

En el siguiente fragmento de código se ha utilizado una imagen cuyas **dimensiones** exceden el tamaño de la ventana, por lo que se han **ajustado al tamaño del frame maximizado**.

```
public void paint(Graphics g){
        super.paint(g);
        Toolkit t = Toolkit.getDefaultToolkit();
        Image i = t.getImage(“…/imagen.jpg”);
        int ancho = (int)(t.getScreenSize().getWidth());
        int alt = (int)(t.getScreenSize().getHeight());
        g.drawImage(i, 0, 0, ancho, alt, this);
}
```

DESARROLLO DE INTERFACES

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Escuchadores de eventos en componentes visuales

Los eventos permiten que el usuario pueda establecer una «**comunicación**» con cualquier aplicación para que esta se ajuste a las decisiones del usuario en lo que respecta a su recorrido por un sitio web, aplicación o herramienta.

Funcionalidades de lenguajes dinámicos

Reflexión: **recuperar y modificar** de forma dinámica diferentes **datos** relativos a la **estructura de un objeto**: los **métodos** y **propiedades** de la clase, los **constructores**, las **interfaces** o el **nombre** original **de la clase**, entre otros.

Introspección: permiten a entornos visuales de diseño **tomar de forma dinámica** todos los **métodos**, **propiedades** o **eventos** asociados a un **componente**, que se colocan sobre el lienzo de diseño simplemente arrastrando y soltando.

La **introspección** de los componentes visuales **requiere de la reflexión**.

Ambas propiedades son dos **características clave** en el diseño de **JavaBean**.

Persistencia del componente

Permite que el **estado de una determinada clase no varíe** y es posible a través de la **serialización**. Cuando se crea un **nuevo componente**, será necesario **implementar** alguna de las **interfaces** siguientes en función del **tipo de serialización** escogida:

- Serialización **automática**. Utiliza la interfaz: **java.io.Serializable.**
- Serialización **programada**. Utiliza la interfaz: **java.io.Externalizable**.

Clases de eventos

La programación basada en eventos es la clave de la **iteración** entre el usuario y una interfaz. Este tipo de programación podría **dividirse** en **dos** grandes **bloques**:

- la **detección** de los eventos.
- las **acciones** asociadas a su tratamiento.

En función del **origen del evento** (dónde se haya producido), diferenciamos entre:

• Eventos **internos**. Este tipo de eventos está producido por el **propio sistema**.

• Eventos **externos**. Los eventos externos son aquellos producidos por el **usuario**, habitualmente, a través del teclado o del puntero del ratón.

Los **objetos** que definen todos los **eventos** que se verán en este tema se basan en las siguientes clases:

Clase	Descripción
EventObject	Derivan **TODOS** los eventos.
MouseEvent	Acción del **ratón** sobre el componente.
Component-Event	**Cambio de un componente (**tamaño, posición...).
ContainerEvent	**Añadir o eliminar componente** sobre un objeto de tipo **Container.**
WindowsEvent	**Variación en una ventana (**apertura o cierre, cambio de tamaño...).
ActionEvent	**Acción sobre un componente**. De los más **comunes**, modela acciones como la pulsación sobre un botón o el check en un menú de selección.

Componentes

Los **componentes** presentan habitualmente un tipo de **evento asociado**.

No es lo mismo el **tipo de detección** asociado a un **botón** o a la pulsación de una **tecla**, que la forma de detección de la apertura o cierre de una **ventana**.

Asociación del evento al componente

Paso 1: crear un componente:

```
JButton btnInicializar = new JButton("Inicializar");
```

Paso 2: añadir al JPanel:

```
contentPane.add(btnInicializar);
```

Paso 3: añadir el escuchador:

```
btnInicializar.addActionListener(new ActionListener() {...}
```

Paso 4: programar la acción:

```
@Override
public void actionPerformed(ActionEvent e) {...}
```

En la siguiente tabla, se muestran los componentes más habituales y el tipo de evento asociado a estos.

Nombre componente	Nombre evento	Descripción del evento
JTextField	**ActionEvent**	Detecta la **pulsación** de la tecla **Enter** tras completar un **campo de texto**.
JButton	**ActionEvent**	Detecta la **pulsación** sobre un componente de tipo **botón**.
JComboBox	**ActionEventItemEvent**	Se detecta la **selección** de uno de los valores del **menú**.
JCheckBox	**ActionEventItemEvent**	Se detecta el **marcado** de una de las **celdas** de selección.
JTextComponent	**TextEvent**	Se produce un **cambio** en el **texto**.
JScrollBar	**AdjustmentEvent**	Detecta el **movimiento** de la barra de desplazamiento (**scroll**).

Listeners

Los listeners o escuchadores quedan a la espera («escuchando») si ese componente produce un evento. Si este se produce, se ejecutan las acciones asociadas a tal ocurrencia. Todo **evento requiere de un listener** que controle su **activación**.

Tipos de listeners asociados al tipo de evento al que corresponden:

Un mismo **tipo de escuchador** (**listener**) puede estar presente en **varios eventos** y componentes diferentes, aunque, normalmente, presentan un **comportamiento muy similar**.

KeyListener

Al pulsar cualquier **tecla**. Se contemplan varios **tipos de pulsaciones**, cada uno de los cuales presentará un método de control propio. Se implementan los **eventos ActionEvent**.

- **KeyPressed**: Se produce al **pulsar** la tecla.
- **KeyTyped**: Se produce al **pulsar y soltar** la tecla.
- **KeyReleased**: Se produce al **soltar** una tecla.

ActionListener

Pulsación (ratón o tecla) sobre un componente. Está presente **en varios tipos de elementos** y es uno de los escuchadores más **comunes**.

La detección tiene lugar ante **dos tipos de acciones**:

- ➔ Pulsación sobre el componente con la tecla **Enter**, siempre que el **foco** esté sobre el elemento;
- ➔ Pulsación sobre el componente con el puntero del **ratón**.

Estos **componentes** implementan los **eventos** de tipo **ActionEvent**:

Componente asociado a ActionListener

- **JButton**: Al hacer **clic** sobre el botón o pulsar la tecla **Enter** con el **foco** situado sobre el componente.
- **JtextField**: Al pulsar la tecla **Enter** con el **foco** situado sobre la **caja de texto**.
- **Jmenultem**: Al **seleccionar** alguna **opción** del componente menú.
- **Jlist**: Al hacer **doble clic** sobre uno de los **elementos** del componente **lista**.

MouseListener

Este evento se produce al hacer **clic** con el ratón sobre algún **componente**. Es posible **diferenciar** entre distintos tipos de **pulsaciones** y asociar a cada una de ellas una **acción diferente**.

Estos **componentes** implementan los **eventos** de tipo **MouseEvent**:

Componente asociado a **MouseListener**

- **mouseClicked**: Se produce al **pulsar y soltar** con el puntero del ratón sobre el componente.
- **mouseEntered**: Se produce al **acceder** a un componente utilizando el puntero del ratón.
- **mouseExited**: Se produce al **salir** de un componente utilizando el puntero del ratón.
- **mousePressed**: Se produce al **presionar** sobre el componente con el puntero.
- **mouseReleased**: Se produce al **soltar** el puntero del ratón.

MouseMotionListener

Este evento se produce ante la detección del **movimiento** del ratón.

Componente asociado a MouseMotionListener

- **mouseMoved**: Se produce al **mover** sobre un componente el **puntero** del ratón.
- **mouseDragged**: Se produce al **arrastrar un elemento** haciendo clic previamente sobre él.

FocusListener

Este evento se produce cuando un elemento está **seleccionado** o deja de estarlo, es decir, al tener el **foco** sobre el componente o dejar de tenerlo.

Se implementan objetos de la clase de **eventos FocusEvent**.

Diferencia entre eventos y escuchadores:

- ***Evento*** *es hacer que* ***algo suceda****, que los programas cobren vida.*
- ***Escuchador*** *o listener es el encargado de* ***escuchar los eventos*** *que suceden. Gracias a ello se logra que suceda lo que queramos cuando se detecta que ha ocurrido el evento deseado. Los listeners se encargarán de* ***controlar los eventos****,* ***esperando*** *a que el evento se produzca para realizar una serie de acciones.* ***Según el evento****, necesitaremos* ***un listener u otro*** *que lo controle.*

Métodos

Cada uno de los **eventos** utilizará un **método** para el **tratamiento** del mismo.

Tras enlazar al escuchador con la ocurrencia de un evento, será necesario ejecutar un método u otro **en función del tipo de evento** asociado.

Relación entre el Listener y el método propio de cada evento

ActionListener:

public void **actionPerformed**(ActionEvent e)

KeyListener:

public void **keyPressed**(KeyEvent e)

public void **keyTyped**(KeyEvent e)

public void **keyReleased**(KeyEvent e)

MouseEntered

public void **focusGained**(FocusEvent e)

public void **lostGained**(FocusEvent e)

MouseListener

public void **mouseClicked**(MouseEvent e)

public void **mouseEntered**(MouseEvent e)

public void **mouseExited**(MouseEvent e)

public void **mousePressed**(MouseEvent e)

public void **mouseReleased**(MouseEvent e)

MouseMotionListener

public void **mouseMoved**(MouseEvent e)

public void **mouseDragged**(MouseEvent e)

```
// Ejemplo de uso con evento sobre Button

        JButton btn = new JButton("botón");
        btn.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {...}

        ...}
```

Asociación de acciones a eventos

Creación del listener

Se crea una nueva **instancia del evento** y este se **vincula con el componente** sobre el que va a actuar.

Para que se produzca esta detección, será necesario **enlazar el evento con los escuchadores**, elementos que se encuentren a la espera de que se produzca algún evento (Listeners). Estos elementos son interfaces que implementan un conjunto de métodos.

La implementación puede realizarse:

- **Añadiendo el evento directamente al componente** y, posteriormente, codificando las **acciones** que se van a llevar a cabo (**Sintaxis 1**).
- Modelando primero todo el tratamiento del **evento** y, a continuación, **asociándolo con el componente** sobre el que actúa (**Sintaxis 2**).

Sintaxis 1:

```
nombreComponente.addtipoEventoListener(new tipoEventoListener() {...}
```

Sintaxis 2:

```
tipoEventoListener nombreEvento = new tipoEventoListener (){...}

nombreComponente.addtipoEventoListener(nombreEvento);
```

El valor de **tipoEventoListener** se obtiene escogiéndolo **en función del evento que se vaya a tratar**.

Asociación de la acción al evento

Cuando se activa y vincula un escuchador o listener a un componente y, por tanto, a la ocurrencia de un evento, los componentes **no realizan un filtrado previo de los eventos** para determinar si los manejan o no, sino que los reciben todos. A través de la **asociación de la acción al evento** se determinará **si se maneja el evento** o no.

A continuación, se implementa el **método** bajo el cual se **desarrolla la acción**, que se ejecutará tras la **ocurrencia** del evento.

En el lenguaje de programación Java, cada **evento está asociado a un objeto** de la clase **EventObject** y, por lo tanto, a un método concreto.

Estructura general para la **definición de este método**:

```
public void métodoDeEvento(TipoEvento e){...}
```

Los métodos relativos a cada evento, prestando especial atención a las diferentes casuísticas que presentan algunos eventos, son los estudiados en el apartado 4.

Ejemplo de detección de la pulsación sobre un botón

Se mostrará el mensaje *Hola Mundo* en la misma ventana en la que se encuentra el botón.

En **primer lugar**, se crea una nueva **clase JFrame** y se inserta un **panel JPanel** para ubicar encima el resto de elementos.

Se coloca una **etiqueta** y un **botón**, en la distribución que se desee, desde la vista de diseño. En la parte del código, de forma automática, se habrá generado:

```
// Se coloca una etiqueta en Panel
JLabel lblNewLabel = new JLabel("...");
// Se añade al panel
contentPane.add(lblNewLabel);
// Se crea un nuevo botón y añade al panel
JButton btnNewButton = new JButton("Pulsa aquí");
contentPane.add(btnNewButton);
```

Crea el **código** relativo a la **producción y detección de eventos** para cada componente:

Desde la vista de **diseño**, hacemos doble clic sobre el botón, lo que nos lleva al código, donde ahora aparecen algunas líneas nuevas.

```
	btnNewButton.addActionListener(new ActionListener({
```

Se **implementa** tanto **el** `escuchador` vinculado al botón (ActionListener) como **el** `método` dentro del cual se desarrollan las **acciones desencadenadas** por el evento (actionPerformed), que recibe por **parámetro** un **`ObjectEvent`** de **tipo** `ActionEvent`.

Finalmente, solo quedará colocar el **código** que envía a la **etiqueta de texto** creada en el inicio el **mensaje** «Hola Mundo» y la muestra por pantalla.

```
		public void actionPerformed(ActionEvent e) {
			lblNewLabel.setText("Hola mundo");
		}
	});
```

Pruebas unitarias

El desarrollo de **pruebas** de software es importante en la implementación de **nuevos componentes**, puesto que, de esta forma, se reducen considerablemente los tiempos de desarrollo.

Si se ha **probado y verificado el comportamiento** de un componente, esta acción ya no será necesaria, aunque sí habría que **verificar** cómo sería su **uso en el marco de otro proyecto** que lo utilice.

Las pruebas se diseñan en base al **comportamiento esperado** de un componente, extendiéndose a todas las **casuísticas** posibles.

Las pruebas unitarias son pruebas que se realizan sobre una **funcionalidad concreta**, es decir, sobre una parte del programa, con el objetivo de comprobar si funciona de forma correcta. Para el desarrollo de pruebas unitarias, encontramos el framework de **Java**, **JUnit**. Para **intalarlo**, basta con incorporar algunas **librerías JAR** al **entorno** de desarrollo.

Como resultado de las pruebas, se distinguen dos tipos de escenarios: la **respuesta deseada** y la respuesta **real**. Cuando estas **no coinciden**, será necesario hacer una **revisión completa del código** que se está probando.

Además, las pruebas unitarias deben cumplir las características del conocido como **principio FIRST**.

- **Fast**: **Rápida** ejecución.
- **Isolated**: **Independencia** respecto a otros test.
- **Repeatable:** Se pueda **repetir** en el tiempo.
- **Self-Validating:** **Cada test** debe poder **validar si es correcto** o no a sí mismo.
- **Timely:** ¿Cuándo se deben desarrollar los test? ¿Antes o después de que esté todo implementado? Sabemos que cuesta **hacer primero los test y después la implementación** (TDD: Test-driven development), pero es lo suyo para **centrarnos en lo que realmente se desea** implementar.

JUnit en Eclipse

Para trabajar con JUnit desde Eclipse, lo primero que necesitamos es tener nuestro proyecto creado y añadir la **librería JUnit desde Build Path**. La manera más sencilla es haciendo clic con el botón derecho sobre el proyecto y seleccionando:

➔ *Build Path*

➔ *Add Libraries*

➔ *Junit*

Después, nos aparecerá una pantalla con las versiones de la librería disponibles, y seleccionamos JUnit 5.

Ejemplo clase Prueba unitaria con JUnit5

Crearemos nuestra clase de prueba, que se llamará **como la clase que queremos probar**, pero con la palabra '**test' delante**. Si la clase que queremos probar se llama calculadora.java, la clase de prueba se llamara TestCalculadora.java. Además, esta nueva clase debe **extender de la clase TestCase** e **importar junit.framework.TestCase**.

```
import junit.framework.TestCase;
public class TestCalculadora extends TestCase {...}
```

Métodos de prueba

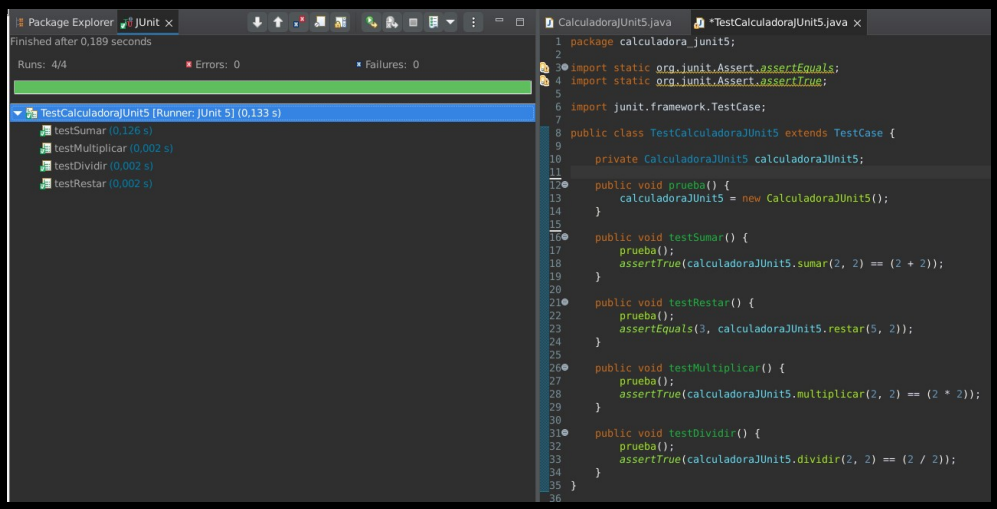
Los **métodos** de esta clase serán **similares** a los de la **clase que se quiere probar**, pero con la palabra '**test' delante**, es decir, si el método original es sumar, el método de prueba será testSumar().

Importante: este método **no devolverá nada**, por lo que se declarará como public **void** TestSumar(). Dentro de cada método, llamaremos al constructor de nuestra clase de prueba y, para **comprobar** si el **resultado obtenido** coincide con el **esperado**, utilizaremos los **métodos assert**. **Los más utilizados** según el tipo de dato que queramos comprobar son:

- **assertTrue**.
- **assertFalse**.
- **assertEquals.**
- **assertNull**.

Comprobación de errores

Por último, comprobaremos los resultados compilando la clase prueba como: **Run As, JUnit Test**. En ese momento, aparecerá un **panel llamado JUnit**, en forma de **árbol**, que mostrará los **resultados correctos** en color verde y los **errores** en rojo.



DESARROLLO DE INTERFACES

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Usabilidad
concepto y normativa

Existe una **extensa normativa** que recoge **reglas y estándares** para la aplicación de la usabilidad en multitud de entornos, definida por la **Organización Internacional de la Normalización** y cuyos fundamentos principales veremos aquí.

Concepto de usabilidad

Facilidad o dificultad de uso de un sitio web, de aplicaciones y herramientas o de cualquier otro entorno software que implique la interacción con un usuario.

Conjunto de parámetros que permiten establecer las **características deseables** sobre una mejor usabilidad. Se trata de diseñar sistemas **eficientes**, **efectivos**, **seguros**, **útiles**, **fáciles** de aprender y fáciles de recordar.

> *Jakob Nielsen: «**Si no lo haces fácil**, los usuarios **se marcharán** de tu web».*

Características de la usabilidad:

• **Eficiencia de uso**: Esta característica hace referencia al **tiempo** que se requiere para **completar una acción** determinada. No resulta muy conveniente que un usuario tarde mucho tiempo en encontrar botones, menús, accesos… O en comprender un funcionamiento excesivamente complejo de la aplicación.

• **Facilidad de aprendizaje**: En relación con la anterior característica, el **tiempo empleado en conocer el funcionamiento** de una aplicación no puede ser demasiado amplio, ya que provocará que el usuario quiera **buscar otras** que **satisfagan** sus necesidades de forma **inmediata**.

• **Retención del tiempo**: En el caso de aplicaciones **utilizadas de manera intermitente**, es aconsejable que el usuario precise de un **menor tiempo de aprendizaje** que la **primera vez que accedió** a la herramienta.

• **Satisfacción**: Una de las características más **subjetivas**, puesto que indica el grado de satisfacción del usuario con respecto al sistema.

• **Tasa de error**: Será deseable que el número de errores cometidos por los usuarios de una aplicación sea lo menor posible. En concreto, aquellos errores que puedan estar **provocados por una excesiva complejidad** de la herramienta.

Principios para el diseño de sistemas interactivos

Se pueden establecer un conjunto de **principios de diseño** relativos a la **interacción** persona-computador. Según el libro **User Engineering Principles for Interactive Systems de Hansen** (1971), estos principios para el diseño de sistemas interactivos son, entre otros:

- **Conocer al usuario**: Es necesario analizar el comportamiento de los '**usuarios tipo**' que van a acceder a la aplicación que se desarrolla, puesto que, en función de estos, se implementarán unas **características** u otras.
- **Minimizar la memorización**: Suele ser aconsejable sustituir las entradas de datos por la **selección de ítems**. Por ejemplo, el uso de nombres aporta una mejor facilidad de uso que el empleo de números. De esta forma, se conseguirá un **comportamiento predecible** y proporcionará **acceso rápido** a la información práctica, y útil, del sistema.
- **Optimizar** las operaciones mediante la **rápida ejecución de operaciones comunes**: La **consistencia** de la interfaz y organizando y **reorganizando la estructura** de la información basándose en la observación del **uso del sistema**.
- **Facilitar buenos mensajes de error**, crear **diseños que eviten** los **errores** más comunes: Haciendo posible **deshacer** acciones realizadas y garantizar la **integridad** del sistema en caso de un fallo de software o hardware (¿backup?).

La **interacción Persona - Ordenador** (**IPO**) o, en inglés, **Human-Computer Interaction** (**HIC**) es «la **disciplina** que estudia el intercambio de información entre las **personas y los ordenadores**», cuyo objetivo es que el **intercambio** y **acceso** a la **información** sea **lo más eficiente posible**.

Normativa

ISO

La **Organización Internacional de la Normalización** (International Organization for **Standardization**, ISO), se encarga de la creación de normas y estándares cuyo objetivo principal es conseguir asegurar que **servicios y productos** presenten ciertos niveles de **calidad, eficiencia y seguridad**.

La ISO define la **usabilidad** como:

> *La capacidad de un producto para ser **entendido**, **aprendido**, **usado** y resultar **atractivo** para el usuario cuando se usa bajo determinadas condiciones. Es decir, la usabilidad no antepone la facilidad de uso al diseño, ni el diseño a la facilidad para comprender su funcionamiento por parte del usuario, será una **combinación de diferentes aspectos clave**, que **podrán variar** en función del **tipo de aplicación** y del **grupo de usuarios** al que va dirigida.*

A continuación, vamos a hacer un recorrido sobre las principales **normas y estándares** enunciados por la ISO en relación a la **consecución de buenos parámetros de usabilidad**, con respecto al diseño y desarrollo de interfaces que favorecen la interacción entre el usuario y la aplicación.

En cuanto al **uso del producto, servicio o herramienta desarrollado**:

- **ISO/IEC 9126-1**: Estándar internacional relativo a la **ingeniería del software** en cuanto a la **calidad** del producto.

  Se distinguen **subestándares**: **modelo** de calidad y **métricas** (de calidad de uso, **externas** e **internas**).

- **ISO/IEC 9241**: Se trata de una **guía de usabilidad** donde se recogen los **beneficios** relativos a las medidas de usabilidad evaluadas. En el siguiente apartado, la analizaremos más en detalle.

En cuanto al **diseño de la interfaz y la interacción** generado entre el usuario y la aplicación son:

- **ISO/IEC 14915**: Estándar internacional relativo a la **ergonomía** del **software** elaborado con respecto a la interfaz multimedia.

- **IEC TR 61997**: Se recogen un conjunto de **guías de interfaz multimedia** para **usuarios** en equipos de uso general.

Normas y estándares de usabilidad. ISO 9241

El estándar ISO/IEC 9241 es uno **de los más amplios** y recoge múltiples subapartados que organizan de forma clara todas las **normas** relativas a la **usabilidad** en el desarrollo de **sistemas interactivos**.

ISO 9241-**10**: Principios de **diseño de los diálogos** que se han de generar entre el usuario y el sistema.

ISO 9241-**11**: E**rgonomía** de la interacción entre el usuario y el sistema.

ISO 9241-**12**: **Organización y disposición** de la información.

ISO 9241-**13**: **Modelado y diseño** de las **ayudas** implementadas en la aplicación para el usuario.

ISO 9241-**14**: Diseño de los **diálogos** producidos por los diferentes elementos que modelan un **menú**: cómo se han de modelar y qué texto poner en botones, casillas, etc.

ISO 9241-**15**: **Diálogos** de lenguaje de **órdenes**.

ISO 9241-**16**: **Requisitos ergonómicos** para el correcto diseño de aplicaciones para trabajos de **oficina** con pantallas de **visualización de datos**.

ISO 9241-**17**: Diseño de **formularios**.

Otros estándares propios de este tipo de desarrollo:

• **ISO 13407**: **Diseño de aplicaciones e interfaces** en sistemas interactivos, poniendo el foco en el **usuario**.

• **ISO/TR 16982**: **Métodos ergonómicos** presentes en las distintas **fases del ciclo de diseño y desarrollo**.

Diferencia entre usabilidad y accesibilidad

Accesibilidad: *especifica en qué grado un producto, o en el caso del software una interfaz, está* ***disponible*** *para ser usada por el* ***mayor número de personas*** *posible. Facilidad para* ***acceder a una interfaz****. Suele enfocarse a personas con* ***discapacidades*** *o limitaciones en su uso, y en los* ***derechos de uso y acceso****. En realidad, se debe recordar que una interfaz que sea accesible* ***beneficia a todas las personas****.*

Usabilidad: *Grado en que una interfaz puede ser* ***utilizada para lograr objetivos*** *concretos con* ***eficacia, eficiencia y satisfacción*** *en un determinado contexto de utilización. De esta manera una interfaz será usable si se logra que su* ***utilidad*** *sea* ***mayor*** *que el* ***esfuerzo*** *necesario para aprender su manejo y el tiempo que será necesario invertir en su uso (fácil de usar).*

Medidas de usabilidad:
Satisfacción, efectividad y eficiencia

Son **herramienta clave para la evaluación** de este concepto en el desarrollo de interfaces adecuadas para la aplicación y los usuarios a los que va dirigida.

Los **test de usabilidad evaluan:**

- La **facilidad de uso** por parte de un usuario (no experto).
- Que la funcionalidad desarrollada **cumple con la finalidad** de la aplicación. Si el desarrollo resulta intuitivo para una persona, pero no cumple sus expectativas en cuanto al objeto de desarrollo, tampoco estará cumpliendo los criterios de usabilidad.

Los test de usabilidad **no son** herramientas de opinión **subjetiva**, ya que se deben desarrollar de forma exhaustiva para que, de **manera objetiva**, se evalúen todos los **parámetros deseados**.

Por lo tanto, en el diseño de este tipo de test, se deben contemplar ciertas **métricas comunes** que devuelven de manera fiable la usabilidad presenten en el sistema evaluado.

Satisfacción

Es clave para la evaluación de la aplicación. La obtención de un nivel bajo de satisfacción, unido a un buen diseño de cuestionario, permitirá añadir las modificaciones oportunas al sistema.

Métricas de satisfacción:

- ➔ Calificación de **satisfacción** del usuario sobre la aplicación.
- ➔ Frecuencia de **reutilización** de la aplicación.
- ➔ Calificación relativa a la **facilidad** de aprendizaje.
- ➔ Medida de **uso voluntario** de la aplicación.

Efectividad

Grado de éxito de una aplicación, si **cumple con la funcionalidad** para la cual ha sido desarrollada. Este atributo está estrechamente ligado a la facilidad de aprendizaje de la herramienta. Será necesario contemplar distintos escenarios.

Métricas de efectividad:

- ➔ Cantidad de **tareas relevantes completadas** en cada uno de los intentos.
- ➔ Número de acceso a la **documentación**, al **soporte** y a la **ayuda.**
- ➔ Cantidad de **funciones aprendidas**.
- ➔ Número de **usuarios capaces de aprender** las características del producto.
- ➔ Cantidad y tipos de **errores tolerados** por los usuarios.
- ➔ Cantidad o porcentaje de **palabras leídas correctamente**.

Eficiencia

Tiempo que es necesario **para completar una tarea** con el software desarrollado (a más eficiencia más tareas completadas).

Las métricas definidas para este atributo están **basadas**, sobre todo, en el **primero de los intentos**, puesto que, en ese momento, aún no se tiene demasiado conocimiento sobre la aplicación. Si en este intento los **valores son buenos**, en los **restantes ofrecerán grandes resultados**, que se valoran en las métricas de efectividad.

Métricas de eficiencia:

- ➔ Tiempo **productivo**.
- ➔ **Tiempo** para **aprender el funcionamiento**.
- ➔ **Tiempo** requerido en el **primer intento** para completar la **funcionalidad** evaluada.
- ➔ **Eficiencia** relativa al **primer intento**.
- ➔ **Errores persistentes**.
- ➔ Tiempo necesario para **aprender de nuevo** la funcionalidad del producto **pasado un tiempo** desde su anterior uso.

Diseño y realización de pruebas de usabilidad

Requiere de un **algoritmo**.

El **diseño de los test de usabilidad** cumple un papel activo **en el proceso de implementación** de la aplicación.

El flujo no solo se ejecuta una vez, es decir, cuando se completa el primer diseño y se realizan las pruebas y test necesarios, puede extenderse todo lo necesario.

Estos **pasos necesarios** para un desarrollo completo son:

1. Se realiza un **análisis** completo del **perfil** de **usuario** de la aplicación.
2. Se desglosan los diferentes **flujos de trabajo** del usuario de la herramienta (**funcionalidades**).
3. Se diseñan los **parámetros de usabilidad** necesarios.
4. Se comienza a implementar el **diseño de la aplicación**, tanto a nivel de **funcionalidad** como de **interfaz**.
5. Tras este primer desarrollo, se obtienen los llamados **prototipos**, sobre los cuales se comienza a aplicar los **test de usabilidad**, creados **en base a los parámetros** definidos en el punto 3.
6. En base a estos resultados, se irá **rediseñando y readaptando** las diferentes versiones del software, hasta obtener un **resultado final** que puede llevarse a producción, donde será distribuida de la forma pertinente.

Pruebas con expertos (evaluación heurística)

Las **pruebas realizadas por expertos** realizan una inspección del software basada en su **conocimiento** de la aplicación, así como en un listado de todos los **posibles escenarios** que se pueden contemplar.

Las **pruebas realizadas por usuarios** no llevan a cabo una evaluación tan profesional, ya que se basa en otra serie de criterios.

Los **expertos** realizan la conocida **evaluación heurística** o **método de inspección:** analizan toda la aplicación e identifican los **problemas existentes** o algunos **que pudieran ocurrir** antes de llevar a producción una aplicación. El coste de **corrección después** de implantar cualquier software o herramienta es mucho **más elevado** que si se realiza antes.

La **evaluación heurística se divide en dos partes**:

- Evaluación **detalle**: Se realiza un análisis **exhaustivo**.
- Evaluación de **alto nivel**: Se analiza el funcionamiento de forma **general**.

Número de expertos: Suelen ser **cuatro**, que actuarán de forma **independiente**, evaluando cada **funcionalidad** y analizando todos los posibles **problemas** surgidos.

Finalmente, se realiza un **informe de manera conjunta**. Es deseable que estos tengan bastante **experiencia**, tanto en la detección de **errores de usabilidad** como en el manejo de la propia herramienta; de esta forma, **detectarán más problemas** que personal ajeno al producto y al análisis de estos.

Estas pruebas se realizan en **cualquier momento** del proceso de **desarrollo**.

Es aconsejable que se realicen **antes** de las **pruebas con usuarios**, puesto que, cuando se hacen estas, el prototipo tiene que estar lo más avanzado posible para que la **versión probada** por usuarios sea **lo más cercana a la realidad**.

Pruebas con usuarios

Se basan en el análisis y evaluación de una herramienta o aplicación software mediante un grupo de usuarios reales que pueden detectar **errores** que los **expertos no han sido capaces de encontrar**.

Los métodos de test con usuarios se basan en el uso de cuestionarios tipo. Según el Diseño Centrado en el Usuario (DCU), los test de usuario se basan en pruebas que **observan la forma de interacción** de los usuarios con el **producto objeto del test**.

Número de usuarios que participan en este test: Aconsejable **al menos, 15**, para poder garantizar una tasa de **detección cercana al 100%**. La elección de estos debe basarse en los **perfiles** a los que está **dirigida la aplicación**. No tendrá sentido probar una aplicación para la gestión logística de un almacén con un grupo de usuarios que no tienen ninguna vinculación a este tipo de áreas.

Las **pruebas** se realizan **de forma individual** y se deben tener en cuenta todas las observaciones que se tomen, desde la primera toma de contacto, hasta la realización de la prueba completa.

Algunos **criterios de diseño** son:

- Pruebas **razonables**, es decir, que un **usuario real** realizará.
- Pruebas **específicas**, es aconsejable realizar pruebas concretas y **no muy genéricas**.
- Pruebas **factibles**, que **pueden realizarse**, no se trata de un examen que los usuarios no deben superar.
- **Tiempo de realización razonable**.

Tipos de test de usabilidad

Cuatro tipos de test de usabilidad. La elección del más adecuado se hará en base al tipo de estudio:

• Test de uso **pautado**: En este caso, un responsable se encarga de **monitorizar todas las pruebas** que se hacen en base a un listado previamente diseñado sobre un **prototipo muy cercano** a la versión real.

• Test de uso **descontextualizado**: Implementa un proceso similar al anterior, sobre un **prototipo no real**.

• Uso **natural**: Estas pruebas se realizan sobre la **versión final** y con una interacción también similar a la real, **sin ningún moderador** que paute la acciones.

• **Híbridos**: Combinación entre cualquiera de las descritas.

Pautas para el diseño de las pruebas de usabilidad:

1. Se definen los **objetivos** de las pruebas.
2. Se diseña el **formato** y **tipo de datos** que se tomarán y analizarán en el estudio (cuantitativos, cualitativos, mixtos, presencial o en remoto...).
3. Se realiza el **diseño de las pruebas** que se van a realizar.
4. Se determina el **número de personas** que participan en el estudio. Estos se escogen adecuando sus perfiles al **tipo de producto** que se va a evaluar. En el caso de los **expertos**, será deseable que tengan cierta **experiencia** en evaluaciones de usabilidad.
5. Se escogen las **métricas** que se van a recoger en los **test de usabilidad**..
6. Se **implementa** el plan de test: La guía de las pruebas que se van a realizar, el tiempo que se va a destinar a cada una de ellas, etc. Todos los **aspectos relativos** a estas pruebas tienen que estar cuidadosamente escogidos para abarcar el **mayor número posible de casuísticas** y prever cualquier tipo de **contratiempo** que pudiera **entorpecer la satisfacción** del usuario con respecto a la aplicación.

DESARROLLO DE INTERFACES

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Usabilidad
Pautas de diseño

Pautas de diseño de la estructura de la interfaz de usuario:

Una de las principales pautas de diseño que se deben tener en cuenta son las relativas al **diseño de la estructura de la interfaz**, esto hace referencia a la disposición de las ventanas, menús y cuadros de diálogos que son mostrados al usuario del sitio.

En concreto, se pondrá el **foco de atención** en:

- **Punto focal**. El punto de atención debe situarse de forma estratégica para **no impedir** al usuario su **interoperabilidad** con la interfaz.
- **Estructura y consistencia de ventanas**. Esta característica manifiesta la necesidad de mantener cierta consistencia en la estructura de diseño de las ventanas, siendo aconsejable que aquellas que se destinen al **mismo fin** o similar sea **parecidas**. Incluso es deseable que la consistencia de la estructura se mantenga **a lo largo de toda la aplicación**. Por ejemplo, si el menú de opciones se encuentra en la parte superior, será aconsejable que se mantenga ahí durante todo el diseño, y no que este cambie de sitio en cada nueva ventana.
- **Relación entre elementos**. Esta relación entre elementos debe construirse de forma **coherente**.
- **Legibilidad y flujo entre elementos**. El diseño de la interfaz debe construirse de forma legible para el usuario, los elementos se situarán de tal forma que resulten **fáciles de localizar** y de realizar su **lectura**.

Atajos de teclado

Los atajos de teclado permiten **reducir el tiempo de acceso** y puesta en **ejecución** de determinadas **acciones**.

Habitualmente, los atajos se suelen asociara a aquellas tareas que se utilizan de forma más frecuente.

Los eventos realizados con el **ratón** suelen ser **sustituidos** por ciertas combinaciones de **teclas** que **agilizan** la acción.

Por ejemplo, es común que para copiar y pegar un texto se acuda a los conocidos atajos de teclado, Ctrl+C, y para pegarlo, Ctrl+V (estas combinaciones dependerán del sistema operativo). ¿Por qué se utilizan estos atajos? Porque en acciones muy comunes como estas, resultan más ágil hacerlo así que pulsando sobre el menú correspondiente, buscar la opción de copiar y luego repetir el proceso para realizar el pegado.

Menús

El correcto diseño de un menú debe permitir, fundamentalmente, la **correcta navegación** dentro de una aplicación mostrando **todas las condiciones** posibles y permitiendo al usuario **seleccionar las diferentes acciones** mostradas en este menú. Para el diseño de estos, los hitos principales son:

- Se debe mostrar el **título** del menú.
- Se muestran las **opciones** y la **acción asociada**.

El uso de menús supone múltiples ventajas como son la posibilidad de una **navegación rápida e intuitiva** del usuario por toda la aplicación; además, estos permiten mantener una **vista despejada de la aplicación** al **mostrarse y recogerse** a voluntad del usuario.

De forma habitual, los menús aparecen siempre en la **misma zona** de la aplicación, usualmente en la parte **superior** de las aplicaciones. Cuando se pulsa sobre ellos con el ratón o utilizando la combinación de teclas oportunas, es **posible navegar** sobre cada uno de los **títulos** que definen **cada menú y los desplegables** que se abren.

En este tipo de menús, es habitual encontrar la **apertura de combinaciones en cascada**, es decir, **submenús recogidos** dentro de los menús centrales y que **agrupan más acciones**.

También, es posible distinguir entre **menús contextuales** o emergentes, los cuales aparecen normalmente al seleccionar algún objeto y pulsar sobre ellos utilizando el **botón secundario del ratón**. El diseño de este tipo de menús debe tener en cuenta las siguientes **premisas**:

- **No** se aconseja el uso de **menús en cascada**.
- Tampoco es aconsejable que el **número de acciones** recogidas en este tipo de menús sea demasiado grande. Es habitual encontrar **entre 7 y 10 elementos**.
- Las tareas recogidas en estos menús deberán **aparecer también en otro sitio**, puesto que puede que el acceso a los mismos **no sea del todo intuitivo** para todos los usuarios de la aplicación.

Ventanas

La creación de ventanas supone un aspecto clave en el desarrollo de cualquier aplicación, ahora bien, el diseño de estas debe estar **correctamente escogido y desarrollado**, así como el **número de ventanas** totales o el sistema de **apertura y cierre** de las mismas.

Los usuarios deben **controlar diferentes aspectos** de ellas: poder abrirlas, cerrarlas, modificar su tamaño... y **si esto no es posible**, debe existir una **justificación lógica** en su diseño.

La selección de un mal diseño donde **se despliegan ventanas** constantemente y **no se cierran** puede disminuir la usabilidad de la aplicación, ya que devolverán una **interfaz desordenada** que **complicará la navegación** en la aplicación.

Cuadros de diálogo

Permiten la comunicación activa entre una interfaz y el usuario, puesto que a través de **cajas de texto emergentes** se pueden mostrar **mensajes importantes** del sistema sobre los que el usuario puede **realizar pequeñas acciones** como respuesta al mensaje.

En cuanto a las pautas de diseño relativas a estos elementos, se ha de poner especial énfasis en la terminología utilizada en los mensajes, pues estos han de ser **activos y positivos**, y se deben adaptar a las **posibles variantes culturales** en las que se despliega la aplicación. Se aconseja que los mensajes **describan claramente el mensaje**, evitando dar por sentada la información.

Iconos y colores

Las pautas de diseño relativas al aspecto de una interfaz de usuario se centran en **elementos esenciales del diseño más visual** de una aplicación: los **colores**, el tipo de **fuente**, los **iconos** y la **distribución** de los elementos.

El diseño de todos estos elementos debe **facilitar y mejorar la usabilidad** de la aplicación. En muchos casos, una mala elección de este tipo de componentes convierte una gran aplicación en un fracaso.

Iconos

Los iconos permiten asociar acciones a un determinado **objeto** que suele utilizarse **como acceso directo** a la acción que representa, agilizando así la interacción con la propia aplicación.

Ahora bien, su diseño debe ser **representativo de la acción** que enlaza, pero, además, este diseño debe ser todo lo **sencillo** posible, puesto que al mostrarlo sobre una pantalla de forma reducida, puede no verse de manera óptima.

Colores

La elección de una **buena paleta de colores** contribuye de forma decisiva en la experiencia de uso de cualquier aplicación. Los colores se han de escoger de forma adecuada para la **comunicación del mensaje** que se desea transmitir, por ejemplo, si se va a diseñar una aplicación que quiere transmitir un mensaje de calma, no será demasiado aconsejable escoger una paleta de colores extravagantes que pueden transmitir el mensaje contrario.

Ejemplos de **psicología del color** aplicados al diseño de interfaces:

Azul: Tranquilidad, salud...

Negro: Elegancia, perfección...

Verde: Naturaleza, frescura...

Rojo: Pasión, peligro...

Naranja: Confianza, calidez...

Rosa: Dulzura, infancia...

Además, los colores pueden contribuir a **destacar** diferentes elementos de la aplicación, poniendo el **foco** en los más importantes. De esta manera, se aumentará la **eficiencia y velocidad de uso** de la herramienta.

No se debe **abusar del número de colores** ni de un uso desmesurado del color, puesto que esto puede molestar al usuario, creando incluso confusión a la hora de utilizar la aplicación.

Diferencia entre *user experience* y *user interface:*

Experiencia de usuario *(UX,* ***User Experience****):*

Se refiere a la ***participación con un producto*** *digital o físico.*

Interfaz de usuario *(UI,* ***User Interface****):*

Es el ***medio con el que interactúa*** *el usuario, los puntos de contacto entre el usuario y el producto es la propia interfaz por lo que la usabilidad de ésta será determinante en la experiencia de usuario.*

Roles profesionales en usabilidad de interfaces*:*

UX Designer*:*

Se dedican a ***diseñar y desarrollar experiencias*** *significativas para los usuarios a través de procesos relacionados con la* ***marca, funcionalidad y comunicación*** *de una empresa o entidad.*

Arquitecto de información*:*

Su función es ***analizar y organizar*** *la información para* ***facilitar*** *el proceso de* ***comprensión*** *por parte del usuario.*

Diseñador de servicios*:*

Encargado de ***planear y distribuir los recursos*** *de la empresa para* ***mejorar*** *la experiencia de* ***comunicación*** *con los usuarios y trabajadores.*

UX Writter*:*

Establecen una ***comunicación clara*** *con el usuario mediante el uso de* ***texto sencillos*** *y que sean fácilmente* ***comprensibles****.*

UX Researcher*:*

Investiga posibilidades de ***mejora de la interfaz*** *en función del* ***tipo de usuario*** *al que va dirigido el producto.*

UI Designer*:*

Se encarga de desarrollar los ***patrones del producto*** *tanto para la parte web como móvil para lograr una* ***consistencia y apariencia común independientemente*** *del tipo de* ***dispositivo*** *que se vaya a usar.*

Por tanto, la usabilidad va más allá de crear pantallas, es una ***contribución conjunta al diseño*** *de una interfaz mediante la* ***comprensión total*** *del producto.*

Fuentes

La tipografía consiste en el tipo o tipos de letras que se utilizarán en el diseño de una interfaz. El uso de **fuentes familiares** mejora en gran medida la **calidad de la lectura**.

La selección de las fuentes debe basarse en criterios de **legibilidad** y que se ajusten de forma adecuada a la **resolución** de la pantalla. En cuanto a los criterios de **diseño**, se destacan:

- **Tamaño**: Este debe ser adecuado para la lectura y **proporcional a la resolución** de la pantalla.
- **Color**. Si el **texto es oscuro** sobre **fondo claro** se facilita la lectura.
- **Estilo**. **No** es conveniente **abusar** de la **negrita** o el **subrayado**, ni **estilos demasiados sobrecargados** que puedan dificultar la lectura del sitio.

Distribución de los elementos

La distribución de los elementos en las **ventanas** de la aplicación o en los **cuadros de diálogos**, entre otros, también requiere de especial atención, puesto que con ella podemos provocar confusión en los usuarios o, por el contrario, hacer más **intuitiva** su experiencia de uso.

Algunas de las **pautas recomendadas** son:

- **Evitar** el uso de elementos y ventanas **superpuestas**.
- Disponer los **elementos** de manera que se **facilite el seguimiento y lectura** de los mismos. Por ejemplo, si se va a seguir una secuencia de pasos, resultará más intuitivo hacerlo de izquierda a derecha y de arriba a abajo.

Una interfaz responsiva es la que muestra correctamente los elementos en pantalla, y no se cortan o no se ven ordenados, o correctamente.

Si en una aplicación de dispositivo móvil hay que poner elementos, intentar ponerlos en las esquinas. Estos elementos hay que intentar ponerlos en lugares que no queden ocultos al usar la aplicación, por ejemplo en la esquina inferior derecha, si ese elemento es muy importante en la acción buscada. Siempre será mejor colocarlos en la parte superior de la app.

Elementos interactivos del interfaz de usuario

Los elementos interactivos son aquellos que permiten la comunicación activa entre interfaz y usuario. Estamos hablando de componentes tales como: los **checkBox**, los **menús desplegables** que permiten seleccionar una opción, los **botones** o los elementos de tipo **radioButton**, entre otros.

Cuadros de texto y etiquetas

Los cuadros de diálogo o ventanas que se utilizan para mostrar un mensaje a través de etiquetas, también, pueden incorporar **cajas** en blanco que permiten **introducir algún dato**.

Las **pautas de diseño** relativas a estos componentes para incrementar la **legibilidad** de los mismos son:

- Se debe añadir un **texto explicativo** que indique **cómo** se ha de completar cada caja de texto.
- Se recomienda **ajustar el tamaño** de las **cajas** al de la **ventana** donde son expuestos.

Botones, checkBox o radioButton

Este tipo de elementos permiten **escoger un valor** o conjunto de estos y **enviarlo a la aplicación** para realizar las acciones oportunas.

En cuanto a su diseño, deben cumplir las siguientes **pautas**:

- Los **títulos** deben ser **intuitivos**.
- Las **acciones** codificadas en cada opción deben quedar suficientemente **comprensibles** para el usuario.
- Las **opciones** deben ser fácilmente **distinguibles** unas de otras y, por tanto, relativamente **rápidas de escoger** y seleccionar.

El conjunto de esas pautas implica que el usuario **no necesite investigar** sobre la función de cada una de las opciones, lo que ralentizaría el uso de la aplicación y disminuiría la satisfacción del usuario con respecto a la misma.

Menús desplegables

Este tipo de elementos también permiten **escoger un valor** o conjunto de estos y **enviarlo a la aplicación** para realizar las acciones oportunas.

En cuanto a su diseño, deben cumplir también ciertas **pautas**:

Una de las más importantes es que el **número de elementos** recogidos en este tipo de menús esté en torno a **diez**, de lo contrario, el usuario se perderá entre las opciones.

La presentación de datos

El diseño de una interfaz incluye **numerosos tipos de recursos** que, junto con el resto de elementos funcionales descritos, constituyen la apariencia global de la aplicación. Por lo tanto, hay que prestar la atención adecuada a la presentación de todos los datos.

Tablas

Las tablas suelen mostrar la información de una forma estructurada enfatizando los datos que en ella se recogen, ahora bien, no se debe abusar del uso de este tipo de elementos, puesto que se correrá el riesgo de trivializarlas. En cuanto a las pautas de **diseño**, debemos priorizar:

- Los datos deben quedar recogidos bajo unas **etiquetas claras y concisas** que no requieran de explicaciones adicionales.
- Debe aparecer siempre el **título de la tabla** y su longitud **no** debe **exceder** las **dos líneas** de texto.
- Es aconsejable utilizar **encabezados de fila o columna**, según proceda, que resuman el contenido de la fila o columna.
- El diseño debe mostrar la **información** de la forma más **clara** posible.

Gráficos

El uso de gráficos también debe hacerse de forma **balanceada** con respecto a la aplicación.

Algunas de las **pautas de diseño** para la selección de este tipo de elemento son:

- El **tamaño** debe adecuarse a las **dimensiones de la pantalla**.
- **No** se debe **abusar** del número de gráficos.
- Es aconsejable utilizar **pocos**, pero que aporten un **valor añadido** a la aplicación.

La secuencia de control de la aplicación

La secuencia de control de la aplicación hace referencia al **conjunto de acciones** que se deben seguir para **completar un objetivo** concreto, es decir, serán estas secuencias las que determinan el **funcionamiento de la aplicación**. El diseño secuencial de acciones debe partir **de lo general a lo particular**.

Para realizar el diseño de estas secuencias, se necesitará desarrollar varios **prototipos** sobre el **funcionamiento** que se pretende conseguir, tratando de encontrar el **equilibro** perfecto entre el **número de pasos óptimos** para completar la acción, haciendo que estos sean lo suficientemente **intuitivos** para todo usuario desde su primera toma de contacto con la aplicación.

Pautas de diseño de las secuencias de control:

- Diseñar y establecer de forma clara el **objetivo principal** asociado a cada uno de los **elementos** de la interfaz.
- Establecer la consecución de cada objetivo con una **secuencia de control válida**.
- **Mostrar y documentar**, para el usuario, la **secuencia establecida** para que pueda implementarla sin problemas.
- En la medida de lo posible, utilizar la **regla** de los **tres clic**, lo que implica que para llegar a cualquier objetivo, sean necesarios, **como máximo**, este número de selección de opciones. Si es muy superior, puede suponer una pérdida de usabilidad para la aplicación.

Para ver esto de forma más clara, utilizaremos el siguiente **ejemplo**:

> si se desea **crear un nuevo documento** con unas características determinadas, en primer lugar, se deberá acudir a la opción del menú que se encarga de las **nuevas creaciones**, a continuación, se selecciona el **tipo de fichero** y la disposición o plantilla que se va a utilizar y, finalmente, se escoge la **ubicación y el nombre** con el que se almacenará.

Todo esto supone una secuencia de acciones que tiene como objetivo final la creación de un nuevo documento, y que se divide en varias acciones, desde la más general hasta la más particular.

Aseguramiento de la información

El aseguramiento de la información aporta **consistencia** a la información que la aplicación va a manejar.

Las **características** claves de este aseguramiento implican:

- **Integridad** de la información.
- **Disponibilidad** de los datos en el momento en el que son requeridos.
- **Confidencialidad** de la información bajo el diseño de los procesos de **autenticación** oportunos.

Además, el aseguramiento de la información no solo se centra en los datos como tal, sino que está presente en los siguientes **ámbitos**:

- **Datos**.
- **Procesos**.
- **Comportamiento**.
- **Sistema de gestión** de la empresa.

Aplicaciones multimedia

La carga de elementos multimedia

El uso de elementos audiovisuales en aplicaciones multimedia implica el uso de componentes tales como vídeos, animaciones o audios. Una correcta selección de estos elementos será muy útil para el desarrollo de una interfaz más usable puesto que el contenido de tipo audiovisual suele **mejorar** la **capacidad de atención** añadiendo **énfasis** a los puntos clave de la aplicación.

Una de las pautas principales de diseño para la incorporación de elementos extra es mostrar una **carga balanceada entre texto e información**, y **elementos multimedia**.

Las principales **consideraciones de diseño** que se deben seguir para la inclusión de este tipo de elementos son:

→ **simplicidad**,

→ **adaptabilidad**,

→ **coherencia**.

Imágenes

El uso de imágenes también debe hacerse **de forma balanceada** con respeto a la aplicación.

Es importante que presenten **buena calidad**, puesto que aportará a la aplicación un mayor grado de detalle, si se escogen fotografías u otro tipo de diseños de mala calidad, puede disminuir el grado de satisfacción del usuario sobre la aplicación.

PAUTAS PARA IMÁGENES

- El **tamaño** debe ajustarse a las **dimensiones de la pantalla**.
- **No** conviene **abusar** de las imágenes sin ningún fin.
- Es aconsejable utilizar **pocas**, pero que aporten un **valor añadido** a la aplicación, por ejemplo, utilizando **diagramas de uso** o funcionamiento que ayuden al usuario a utilizar la herramienta.

Animaciones

Las animaciones son pequeños vídeos compuestos por una **breve secuencia de imágenes** con movimiento que se muestran al usuario. Es muy importante no abusar en su uso con este tipo de elementos, puesto que pueden sobrecargar la aplicación sin aportar ningún valor, pudiendo incluso llegar a restarlo.

Vídeos

Los vídeos son utilizados, sobre todo, para mostrar de forma visual **todo tipo de acciones, procesos o productos**, entre otros.

PAUTAS PARA VÍDEOS

- El **tamaño** debe ajustarse a las **dimensiones y resolución de la pantalla**.
- **No** conviene **abusar** de los vídeos. En este caso, es especialmente importante, puesto que pueden **ralentizar** mucho el **funcionamiento** de la aplicación.
- Añadir los **elementos de control** necesarios para que el usuario pueda manejar la reproducción de un vídeo en función de sus necesidades.