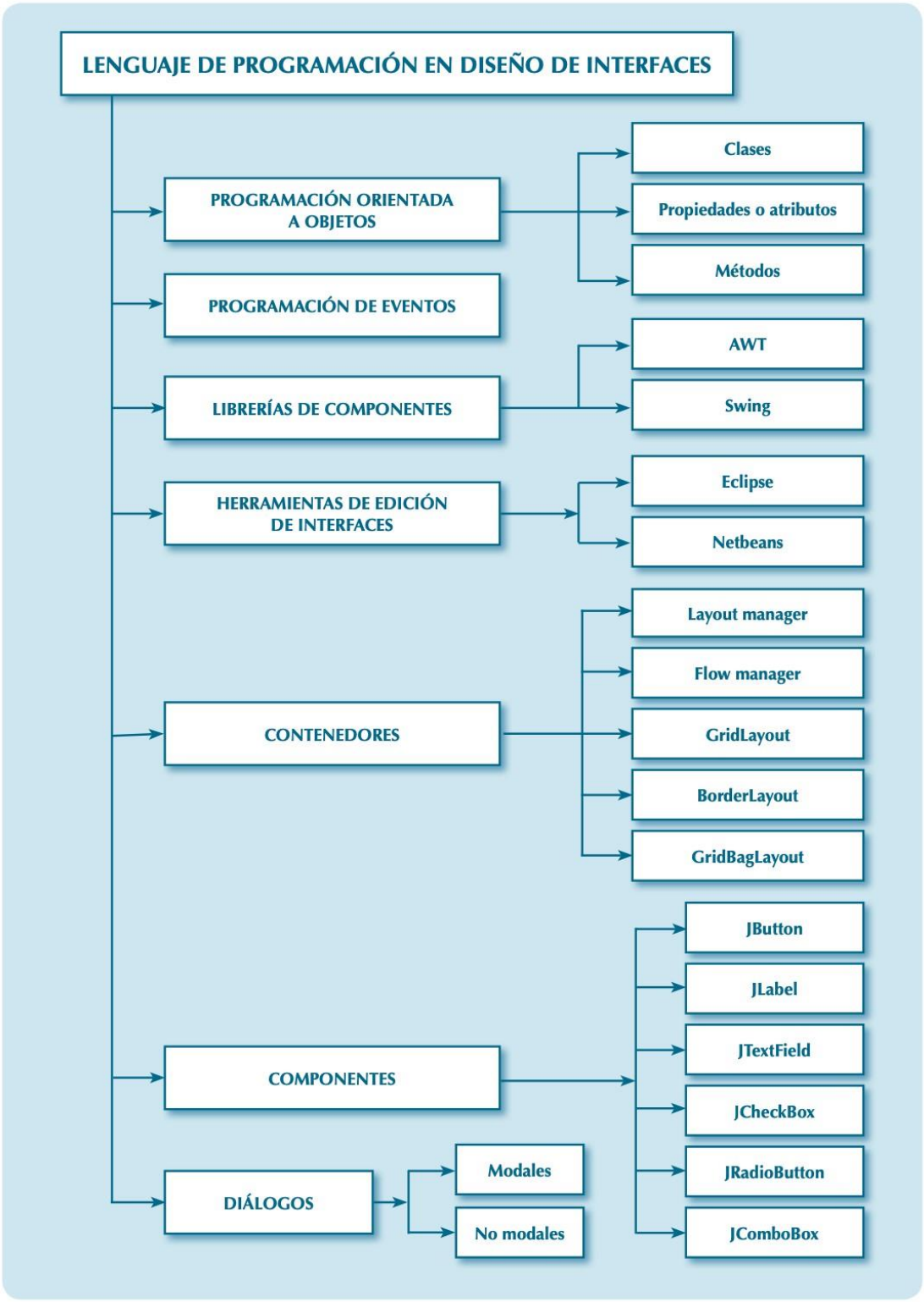
UD1. Lenguaje de programación y confección de interfaces de usuario (RA1)

Objetivos

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

Mapa conceptual



Glosario

Atributo. Estado relacionado con un objeto.

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

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

Evento. Suceso que genera la acción de un objeto.

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

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

Método. Comportamiento relacionado con un objeto.

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

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

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

1.1. Programación orientada a objetos

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

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

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

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

1.1.1. Clases

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

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

Repaso

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

1.1.2. Propiedades o atributos

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

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

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

1.1.3. Métodos

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

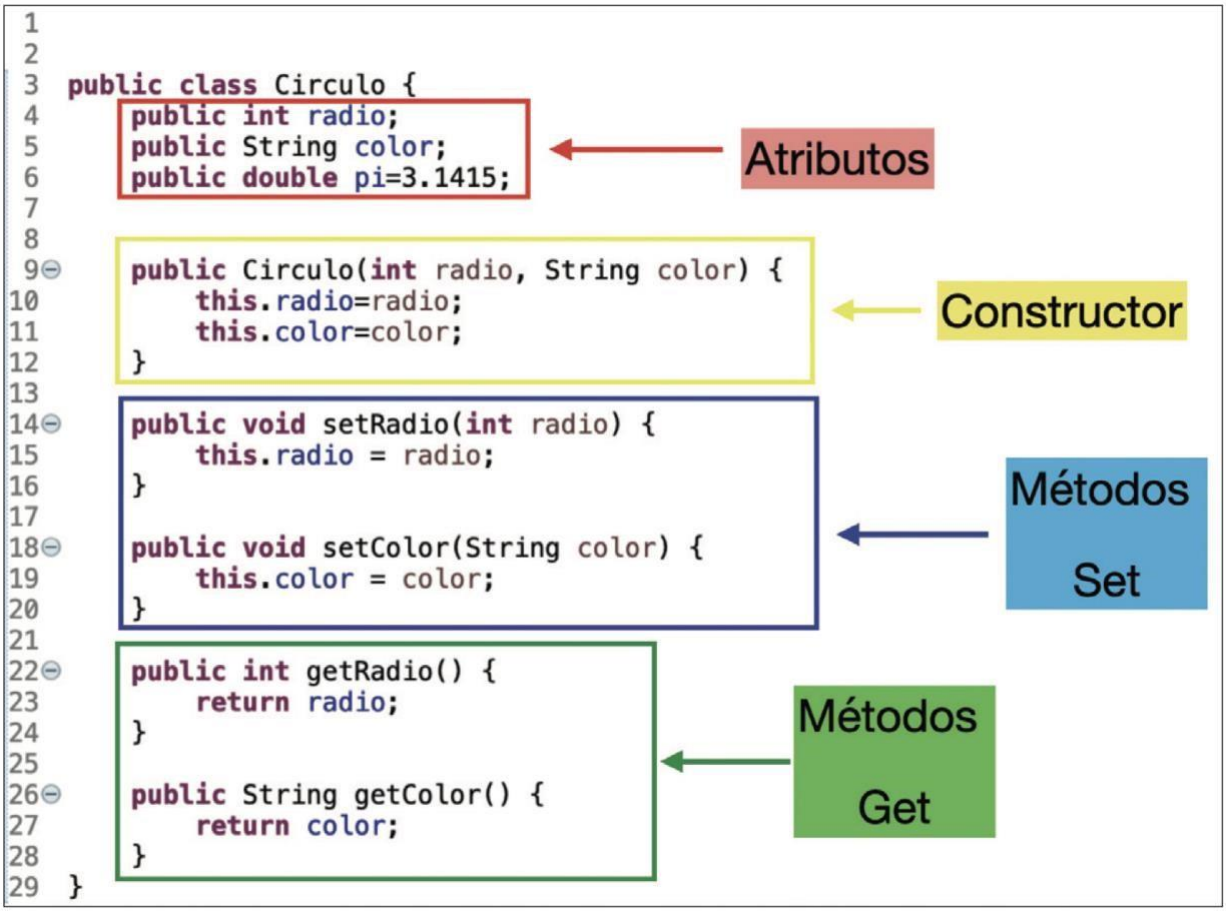


Figura 2.1 Estructura básica clase Java.

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

1.2. Creación de ventanas y programación de eventos

El principal componente para la creación de interfaces visuales es el contenedor principal o ventana. Para la creación de una ventana necesitaremos el componente JFrame de la librería SWING de Java.

Los pasos para crear una ventana serían los siguientes:

Pasos para crear una ventana con componentes y funcionalidades en JavaScript mediante librerías AWT y Swing serían:

1. Importar las siguientes librerías

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

2. Crear el marco principal o ventana:

```
JFrame frame = new JFrame("Nombre del frame”);   //Creación del frame o ventana
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);   //Salir al cerrar ventana
frame.setSize(400, 300);   //Tamaño frame
```

3. Definir cómo se van a distribuir los componentes en pantalla con un layout

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

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

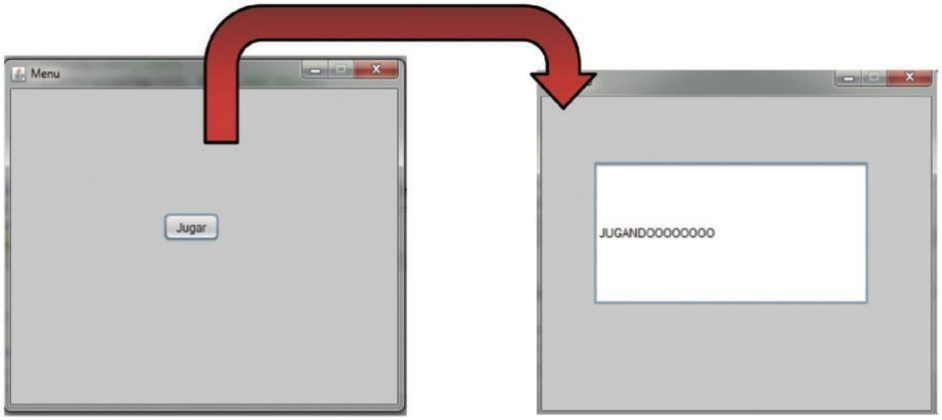
Según la programación orientada a objetos, ¿qué define el comportamiento de un objeto y puede recibir o no argumentos?

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

Figura 2.2 Evento implementado para conexión de JButton.

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

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




```
private void jugarActionPerformed(java.awt.event.ActionEvent evt) {
    JUEGO J1=new JUEGO();
    J1.setVisible(true);
    dispose();
}
```

```
public JUEGO() {
    initComponents();
    setLocationRelativeTo(null);
    setResizable(false);
    setTitle("Juego");
}
```

Figura 2.3 Conexión de ventanas con ActionListener.

Nota:

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

Actividad 1.2

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

1.3. Librerías de componentes

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

1.3.1. AWT

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

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

Figura 2.4 Código para importar librerías Swing y AWT.

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

1.3.2. Swing

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

Figura 2.5 Comparativa de la librería AWT y Swing.

Actividad 1.3

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

1.4. Herramientas de edición de interfaces

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

Recurso web

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

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

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

1.5. Contenedores

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

1.5.1. Layout manager

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

1.5.2. FlowLayout

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

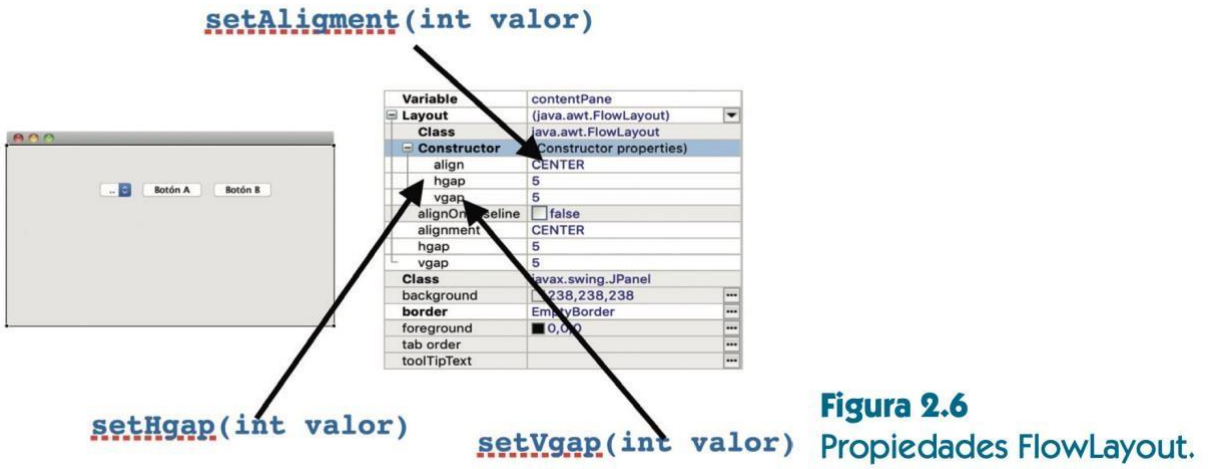


Figura 2.6 Propiedades FlowLayout.

1.5.3. GridLayout

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

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

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

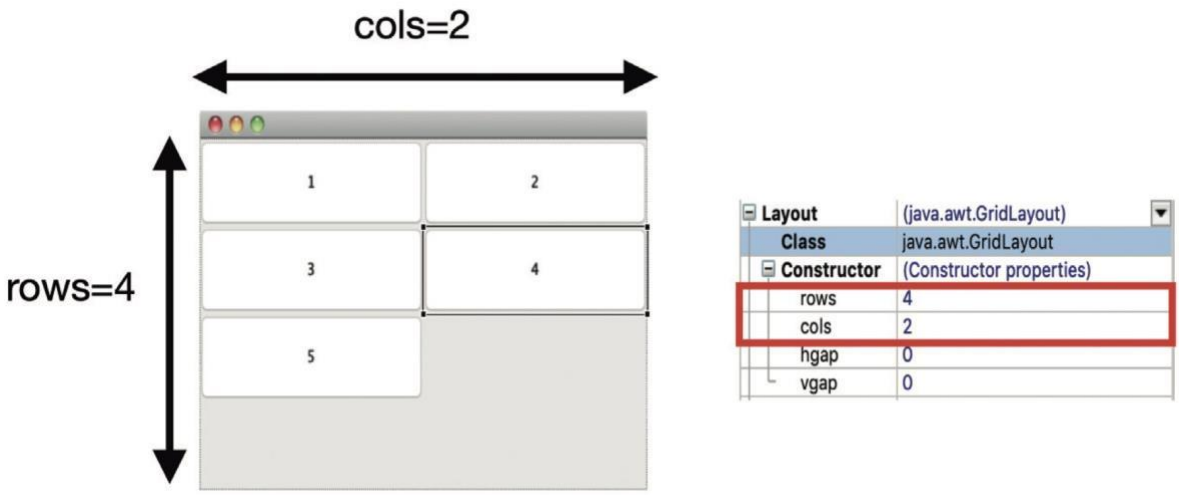


Figura 2.7 Propiedades y ejemplo de botones en GridLayout.

1.5.4. BorderLayout

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

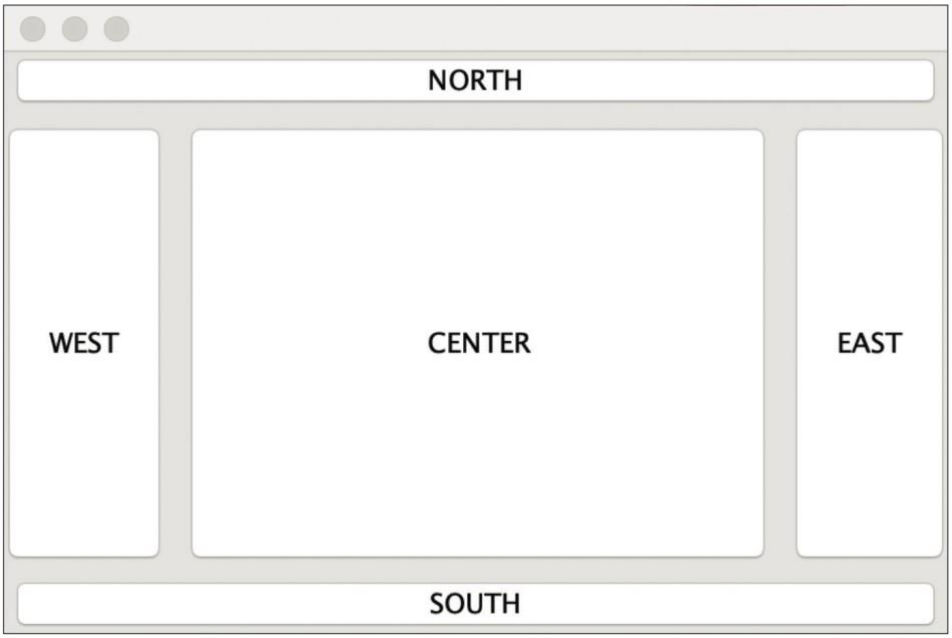


Figura 2.8 Propiedades y ejemplo de botones en BorderLayout.

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

1

.5.5. GridBagLayout

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

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

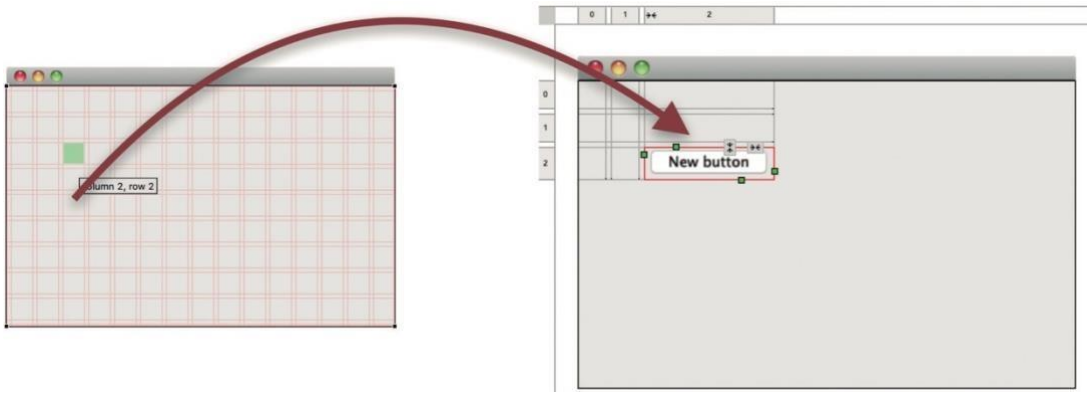


Figura 2.9 Ejemplo botones en GridBagLayout.

1.6. Componentes

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

1.6.1. JButton

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

Cuadro 1.1 Propiedades JButton

background	Color de fondo del botón. Se muestras 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 horizontal y vertical del texto con respecto al botón

1

text	Texto que aparece dentro del botón
icon	Permite cargar una imagen como fondo del botón.

.6.2. JLabel

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

Cuadro 1.2 Propiedades Jlabel

background	Color de fondo de la etiqueta si está habilitada.
enabled	Habilita la etiqueta.
font	Fuente del tipo de letra y tamaño.
foreground	Color del texto si la etiqueta está habilitada.
horizontalAlignment verticalAlignment	Alineación horizontal y vertical 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.

1.6.3. JTextField

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

Propiedades JtextField

background	Color de fondo de la caja de texto.
columns	Tamaño de la caja de texto.
enabled	Habilita el campo de texto.

1

editable	Permite al usuario modificar el contenido.
font	Fuente del tipo de letra y tamaño.
Foreground	Color del texto.
horizontalAlignment	Alineación horizontal del texto.
Text	Texto que aparece al inicio en la caja.

.6.4. JCheckBox

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

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

Figura 1.10 Interfaz ejemplo con JCheckBox y JButton.

1.6.5. JRadioButton

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

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

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

1

1.6.6. JComboBox

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

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

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

1.7. Diálogos

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

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

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

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

Figura 2.11 Ventana creada JDialog.

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

b) ***Diálogos no modales***: sí permitirán que haya tantos JDialog abiertos como se deseen.

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

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

Figura 1.12 Creación de un elemento JDialog de tipo modal.

Actividades propuestas

1.4. ¿En qué casos crees que es más conveniente que las aplicaciones utilicen JDialog de tipo modal o no modal?

1.5. Indica si el siguiente código es correcto en el caso de querer utilizar un diálogo de tipo no modal:

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

Resumen

- El uso de interfaces gráficas da lugar a una comunicación entre usuario y aplicación que se basa en un diseño accesible y útil. Actualmente, existen en el mercado herra- mientas de desarrollo como Eclipse, que permite la implementación del código o el uso de vistas de diseño para el proceso de diseño y creación de interfaces.
- El paradigma de la programación orientada a objetos permite manipular objetos comocmetáfora para simular entidades reales. Estas instancias son entidades que tienen un determinado estado, comportamiento e identidad.
- Una interfaz puede presentar más de una ventana; en función de la finalidad de la misma encontramos JFrame y JDialog. La segunda establece los llamados diálogos modales o no modales, elementos clave en el desarrollo de interfaces. Dentro de una interfaz, la distribución de los elementos se establece utilizando disposiciones de tipo layout, los cuales permiten colocar los elementos en un sitio o en otro.
- La librería Swing sirve para programar todo tipo de componentes visuales como bo- tones, etiquetas, menús desplegables o casillas de verificación, entre muchos otros. Por otro lado, Swing es una extensión para AWT, un kit de herramientas de widgets utilizados para el desarrollo de interfaces gráficas en Java.

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

  JButton
  JRadioButton
  JCheckBox.
  JLabel.
  JComboBox

- Por otro lado, tener presentes las diferencias existentes entre JFrame y JDialog es imprescindible para poder diferenciar en qué casos se utiliza uno u otro en función de la utilidad que se desea obtener de la interfaz.

- El análisis de eventos realizados es necesario para establecer la acción asociada ante la pulsación de un botón, por ejemplo, crear la conexión necesaria entre dos ventanas, ya sean de tipo JFrame o JDialog, bien desde la vista de diseño o desde el código directamente.

UD3. Creación de componentes visuales en diseño de interfaces (RA3)

Glosario

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

Get. Método que permite analizar el contenido de una propiedad o atributo.

Introspección. Características de los entornos visuales de diseño que permite a estas herramientas tomar de forma dinámica todos los métodos, propiedades o eventos asociados a un componente.

Paquete. Agrupación que contiene lo necesario para desplegar una aplicación, que está compuesto de ficheros ejecutables, elementos multimedia, así como librerías y bibliotecas.

POC. Programación basada en componentes.

Programación basada en componentes. Metodología de programación basada en el uso de elementos reutilizables.

Propiedad. Definen los datos públicos que forman la apariencia y comportamiento del objeto. Pueden modificar su valor a través de los métodos que definen el comportamiento de un componente.

Reflexión. Característica que permite recuperar y modificar de forma dinámica diferentes datos relativos a la estructura de un objeto.

Set. Método que permite modificar el valor de una propiedad o atributo.

3.1. Componentes visuales

El desarrollo de aplicaciones informáticas requiere de una interacción constante entre el usuario y la interfaz. Los elementos visuales que permiten la comunicación entre el usuario y la aplicación son los conocidos como componentes visuales.

En este tema se profundizará acerca de cuáles son los componentes más utilizados, cómo se puede interactuar con ellos y cuáles son las propiedades y atributos de cada uno de ellos.

3.1.1. Concepto de componente

Un componente es un módulo de código ya implementado y reutilizable que puede interactuar con otros componentes software a través de las interfaces de comunicación.

Nota:
La metodología de programación basada en el uso de elementos reutilizables recibe el nombre de programación basada en componentes [POC].

Los componentes permiten la implementación de sistemas utilizando componentes ya desarrollados y, por tanto, probados, lo que conlleva a una notable reducción del tiempo de implementación y los costes asociados. Para conseguir una reutilización eficiente del software es necesario que esté definido de la manera más generalizada posible para poder implementar múltiples versiones modificadas.

A modo de resumen, de todos los tipos de componentes visuales que presenta la librería Jswing se podrá tomar cualquiera de ellos para desarrollar uno nuevo desde cero.

3.1.2. Propiedades y 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. Por ejemplo, si hablamos de un componente tipo JButton, una de sus propiedades será font, la fuente del texto que aparece dentro del elemento, y este valor podrá ser consultado o modificado.

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.

Hay tres tipos de ámbitos de propiedades:

- **Ámbito público**: una propiedad de ámbito público puede ser utilizada desde cualquier parte de la aplicación.

- **Ámbito privado**: una propiedad de tipo privada solo es accesible desde la clase donde se ha creado.

- **Ámbito estático**: pueden ser utilizadas sin la necesidad de crear una instancia del objeto al que está referida.

Nota:
Los atributos, aunque son similares a las propiedades, se utilizan para almacenar los datos internos y de uso privado de una clase u objeto.

Se distinguen principalmente dos tipos de propiedades: **simples** e **indexadas**.

a) Las propiedades simples son aquellas que representan solo un valor. Es el caso de los atributos sencillos como los de tipo String, int o boolean, entre otros.

b) Las propiedades indexadas son aquellas que representan un conjunto de valores en forma de array.

Ejemplo:

Cuando hablamos de un elemento tipo CombaBox, el listado de valores que se muestran en el menú se recogen dentro de una array, por lo tanto, esta propiedad sería de tipo indexada.

```
JComboBox comboBox = new JComboBox();
comboBox.addItem(“primero”); comboBox.addItem(“segundo”);
comboBox.addItem(“tercero”) ;
comboBox.addItem(“cuarto”);
```

3.2. Eventos

La clave de la interacción entre el usuario y una interfaz es la inclusión de eventos; sin estos solo tendríamos textos, imágenes o cualquier otro elemento estático. Este tipo de programación podría dividirse en dos grandes bloques: la detección de los eventos y las acciones asociadas a su respuesta.

En función del origen del evento, es decir, en función de dónde se ha producido, diferenciamos entre eventos internos y externos. Por un lado están los producidos por el propio sistema y por otro los producidos por el usuario.

Los objetos que definen todos los eventos se basan en las siguientes clases.

Cuadro 3.1

Tipos de eventos asociados a objetos

Clase	Descripción
EventObject	Clase principal de la que derivan TODOS los eventos.
MouseEvent	Habilita el campo de texto.
ComponentEvent	Eventos relacionados con el cambio de un componente, de tamaño, posición...
ContainerEvent	Evento producido al añadir o eliminar componente sobre un objeto de tipo Container.
WindowsEvent	Este tipo de eventos se produce cuando una ventana ha sufrido algún tipo de variación, desde su apertura o cierre hasta el cambio de tamaño.
ActionEvent	Evento que se produce al detectarse la acción sobre un componente. Es uno de los más comunes, puesto que modela acciones tales como la pulsación sobre un botón o el check en un menú de selección.

3.2.1. Componente y eventos

Los componentes utilizados para el desarrollo de interfaces normalmente tienen un evento asociado, por ejemplo, no es lo mismo el tipo de detección asociado a un botón o una pulsación de una tecla, que la forma de detección de la apertura de una nueva ventana en una interfaz.

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

Cuadro 3.2
Nombre de componente y nombre de evento asociado

Clase	Nombre evento	Descripción del evento
JTextField	ActionEvent	Detecta la pulsación de la tecla Enter tras completar un campo de texto.
JButton	ActionEvent ItemEvent	Detecta la pulsación sobre un componente de tipo botón.
JComboBox	ActionEvent ItemEvent	Se detecta la selección de uno de los valores del menú.

JCheckBox	ActionEvent ItemEvent	Se detecta el marcado de una de las celdas de selección
JTextComoponent	TextEvent	Se produce un cambio en el texto.
JScrollBar	AdjustmentEvent	Detecta el movimiento de la barra de desplazamiento (scroll).

3.2.2. Listeners

Los listeners o escuchadores permiten detectar la ocurrencia de los eventos, se podría decir que cuando estos se definen y activan quedan a la espera ("escuchando") si se 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.

A continuación, se verán todos los tipos de listeners asociados al tipo de evento al que corresponden. Como se puede ver, un mismo tipo de escuchador puede estar presente en varios eventos y componentes diferentes, aunque normalmente presentan un comportamiento muy similar.

A) KeyListener

Este evento se detecta al pulsar cualquier tecla. Bajo este escuchador se contemplan varios tipos de pulsaciones, cada uno de los cuales presentará un método de control propio. Se implementan los eventos ActionEvent.

Cuabro 3.3
Eventos asociados a KeyListener

Modo de pulsación	Definición
keyPressed	Se produce al pulsar la tecla.
keyTyped	Se produce al pulsar y soltar la tecla.
keyReleased	Se produce al soltar una tecla.
ContainerEvent	Evento producido al añadir o eliminar componente sobre un objeto de tipo Container.
WindowsEvent	Este tipo de eventos se produce cuando una ventana ha sufrido algún tipo de variación, desde su apertura o cierre hasta el cambio de tamaño.

ActionEvent	Evento que se produce al detectarse la acción sobre un componente. Es uno de los más comunes, puesto que modela acciones tales como la pulsación sobre un botón o el check en un menú de selección.

B) ActionListener

Este evento detecta la pulsación sobre un componente, está presente en varios tipos de elementos siendo uno de los escuchadores más comunes.

La detección tiene lugar ante dos tipos de acciones, la pulsación sobre el componente con la tecla Enter siempre que el foco esté sobre el elemento, o la pulsación con el puntero del ratón sobre el componente. Estos componentes implementan los eventos de tipo ActionEvent.

Cuadro 3.4
Componentes asociados a ActionListener

Componente	Descripción
JButton	Al hacer click 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 click sobre uno de los elementos del componente lista.

C) Mouselistener

Este evento se produce al hacer click con el puntero del 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.

Cuadro 3.5
Eventos asociados a MouseListener

Definición	Modo de pulsación

mouseClicked	Se produce al pulsar y soltar con el puntero del ratón sobre el componente.
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.
mouseEntered	Se produce al acceder a un componente utilizando el puntero del ratón.

D) 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.
Para cualquiera de los casos se implementan objetos de la clase de eventos FocusEvent.

E) MouseMotionListener

Mientras que el caso anterior se detectan las acciones realizadas con el puntero en cuanto a pulsar o soltar los bolmes del ratón, este nuevo evento se produce ante la detección del movimiento del ratón.

CuADRO 3.6
Eventos asociados a MouseMotionListener

Modo de acción	Definición
mouseMoved	Se produce al mover sobre un componente el puntero del ratón.
mouseDragged	Se produce al arrastrar un elemento haciendo click previamente sobre él.

3.2.3. Métodos y eventos

Cada uno de los eventos detallados en los apartados anteriores utilizará un método para el tratamiento de cada evento, es decir, tras enlazar al escuchador con la ocurrencia de un evento, será necesario ejecutar un método u otro en función del tipo de evento asociado.

En la siguiente tabla se muestra la relación entre el Listener y el método propio de cada evento:

CUADRO 3.7

Relación Listener-métodos

Nombre Listener	Métodos
ActionListener	public void actionPerformed(ActionEvent e)
KeyListener	keyPressed public void keyPressed(KeyEvent e)
	keyTyped public void keyTyped(KeyEvent e)
	keyRelease public void keyReleased(KeyEvent e)
FocusListener	Obtención del foco public void focusGained(FocusEvent e)
	Pérdida del foco public void lostGained(FocusEvent e)
MouseListener	mouseClicked public void mouseClicked(MouseEvent e)
	mouseExited public void mouseExited(MouseEvent e)
	mousePressed public void mousePressed(MouseEvent e)
	mouseReleased public void mouseReleased(MouseEvent e)
	mouseEntered public void mouseEntered(MouseEvent e)
MouseMotionListener	mouseMoved public void mouseMoved(MouseEvent e)
	mouseDragged public void mouseDragged(MouseEvent e)

3.2.2. Key Binding

El **Key Binding** hace referencia a una técnica que básicamente permite a cualquier componente (que sea extendido de un JComponent) responder a la pulsación de teclas presionadas por el usuario.

Básicamente el uso de Key Bindings se realiza con dos clases: **InputMap** y **ActionMap**.

La clase **InputMap** nos permite "unir" (binding) o "relacionar" un evento de pulsación de teclas con un Objeto. Mientras que la clase **ActionMap** nos permite relacionar dichos objetos con acciones que se deben realizar.

El parámetro que se pone al obtener el InputMap de un objeto o componente indica cuándo debe "accionarse". Normalmente se utilizan 3 condiciones:

- JComponent.WHEN_FOCUSED: Cuando el componente tiene el enfoque del teclado.

- JComponent.WHEN_ANCESTOR_OF_FOCUSED_COMPONENT: Cuando el componente contiene (o es) el componente que tiene el enfoque.

- JComponent.WHEN_IN_FOCUSED_WINDOW: Cuando tu aplicación tiene el enfoque (es decir, cuando no hay otra aplicación enfocada) o bien, contiene el componente que tiene el enfoque.

Por tanto, lo primero que necesitaremos crear para usar esta técnica será crear un objeto de la clase **InputMap** y otro objeto de la clase **ActionMap** de la siguiente manera:

```
InputMap mapaEntrada =
JComponent.getInputMap(WHEN_IN_FOCUSED_WINDOW);
ActionMap mapaAcciones = JComponent.getActionMap();
```

JComponent es el objeto sobre el cual quiero definir el mapa de pulsaciones te teclas y de acciones. Lo habitual es crear una clase extendida del tipo del componente para definir un **Key Binding**, poniendo como atributos private el InputMap y el ActionMap y como método público la creación de los objetos y la asociación de las acciones que veremos a continuación.

El siguiente paso sería capturara el evento de pulsar y soltar para una tecla determinada por un objeto de tipo **char**, esto se hace mediante la clase **KeyStroke** de la siguiente manera:

```
char tecla = 'A'; //indico el carácter de la tecla sobre la que quiero capturar el evento
KeyStroke pulsacionTecla = KeyStroke.getKeyStroke(tecla); //capturo el evento para la tecla
```

A continuación, habría que añadir al objeto de **InputMap** el objeto de **Keystroke** como primer parámetro y como segundo parámetro el objeto de **KeyStroke** convertido a tipo String:

```
mapaEntrada.put(pulsacionTecla, pulsacionTecla.toString());
```

Por último, habría que definir el objeto de la clase **ActionMap** asociando la tecla pulsada con la acción que queremos que haga:

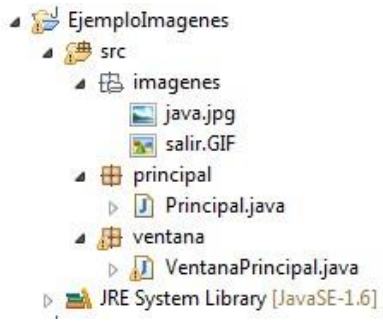
```
mapaAcciones.put(pulsacionTecla.toString(), new AbstractAction() {
        @Override
        public void actionPerformed(ActionEvent e) {
        //Acciones a realizar cuando se pulse la tecla;
        }
});
```

3.3. Vinculación de imágenes a componentes

En este apartado nos enfocaremos en como vincular imagenes a nuestra aplicación. Por ejemplo, tenemos un **JFrame** que contendrá un **JLabel** al cual le asignaremos una imagen de fondo, así como un botón de salir con un icono representativo.

La Aplicación.

Para esta aplicación tan solo utilizaremos 2 clases, la clase principal y la clase Ventana que carga la imagen anterior. Adicionalmente tendremos un paquete de imagenes donde agruparemos todas las imagenes con las que vamos a trabajar. La estructura del proyecto java sería algo parecido a la siguiente figura:



La imagen de Java la alojaremos en un **JLabel** usando para esto el método **setIcon** en el cual instanciaremos un objeto de tipo **ImageIcon** que buscaremos dentro del proyecto en la ruta definida de la siguiente manera:

```
JLabel labelImagen; labelImagen=new
JLabel();
labelImagen.setBounds(50,70,400,330); labelImagen.setIcon(new
ImageIcon(getClass().
   getResource("/imagenes/java.jpg")));
labelImagen.setBorder(javax.swing.BorderFactory.createBevelBorder
    (javax.swing.border.BevelBorder.RAISED));
```

Adicionalmente usando el método **setBorder** le damos un estilo al **jLabel** para que tenga un borde con relieve. Como vemos el método setIcon nos permite alojar una imagen en el componente al que se lo asignemos, la imagen del botón se realizará de la misma manera.

Clase Principal.

En esta clase instanciamos la ventana de nuestra aplicación y hacemos el llamado para que sea visible.

```
package principal;

import ventana.VentanaPrincipal;

/**
*    @author HENAO
*     */
public class Principal {

public static void main(String[] args) {
  /**Declaramos el objeto*/
  VentanaPrincipal miVentanaPrincipal;
  /**Instanciamos el objeto*/
  miVentanaPrincipal= new VentanaPrincipal();
/**Hacemos que se cargue la ventana*/
miVentanaPrincipal.setVisible(true);
 }
}
```

Clase VentanaPrincipal.

Esta clase corresponde a la interfaz de nuestra aplicación.

```
package ventana;
```

```
import java.awt.Container; import
java.awt.event.ActionEvent; import
java.awt.event.ActionListener;

import javax.swing.ImageIcon;
import javax.swing.JButton; import
javax.swing.JFrame; import
javax.swing.JLabel; import
javax.swing.JOptionPane;

public class VentanaPrincipal extends JFrame implements ActionListener {

 private Container contenedor;//declaramos el contenedor
 JButton salir;//declaramos el objeto Boton
 JLabel labelTitulo;//declaramos el objeto Label
 JLabel labelImagen;

 public VentanaPrincipal(){
  /**permite iniciar las propiedades de los componentes*/
iniciarComponentes();
     /**Asigna un titulo a la barra de titulo*/
setTitle("CoDejaVu : Imagenes en Java");
  /**tamaño de la ventana*/
setSize(500,520);
  /**pone la ventana en el Centro de la pantalla*/
setLocationRelativeTo(null);
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 }
 private void iniciarComponentes() {
  contenedor=getContentPane();//instanciamos el contenedor
/**con esto definmos nosotros mismos los tamaños y posicion
*     de los componentes*/
contenedor.setLayout(null);

  /**Propiedades del boton, lo instanciamos, posicionamos y
*     activamos los eventos*/   salir=new
JButton();
  salir.setBounds(390,430,60,30);
  salir.setIcon(new
ImageIcon(getClass().getResource("/imagenes/salir.gif")));
salir.addActionListener(this);

  labelImagen=new JLabel();
  labelImagen.setBounds(50,70,400,330);
labelImagen.setIcon(new ImageIcon(getClass().
getResource("/imagenes/java.jpg")));
  labelImagen.setBorder(javax.swing.BorderFactory.createBevelBorder
        (javax.swing.border.BevelBorder.RAISED));

  /**Propiedades del Label, lo instanciamos, posicionamos y
   * activamos los eventos*/
labelTitulo= new JLabel();
labelTitulo.setText("setIcon");
```

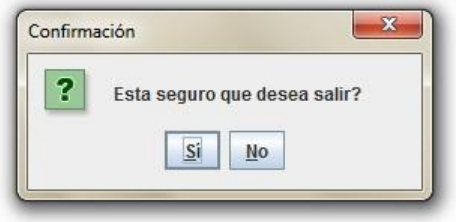
```
labelTitulo.setFont(new java.awt.Font("Comic Sans MS", 0, 28));
```

```
  labelTitulo.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
labelTitulo.setBorder(javax.swing.BorderFactory.createBevelBorder
      (javax.swing.border.BevelBorder.LOWERED));
labelTitulo.setBounds(100, 10, 300, 40);


  /**Agregamos los componentes al Contenedor*/
contenedor.add(labelTitulo);   contenedor.add(salir);
contenedor.add(labelImagen);
 }


 /**Agregamos el evento al momento de llamar la otra ventana*/
 @Override
 public void actionPerformed(ActionEvent evento) {
if (evento.getSource()==salir)
  {
   int respuesta = JOptionPane.showConfirmDialog(this,
     "Esta seguro que desea salir?", "Confirmación",
JOptionPane.YES_NO_OPTION);
   if (respuesta == JOptionPane.YES_NO_OPTION)
   {
    System.exit(0);
   }
  }
 }
}
```

con el botón salir aplicamos uno de los ejemplos mostrados aquí donde por medio de un **JOptionPane** solicitamos que se confirme si deseamos o no salir del sistema.



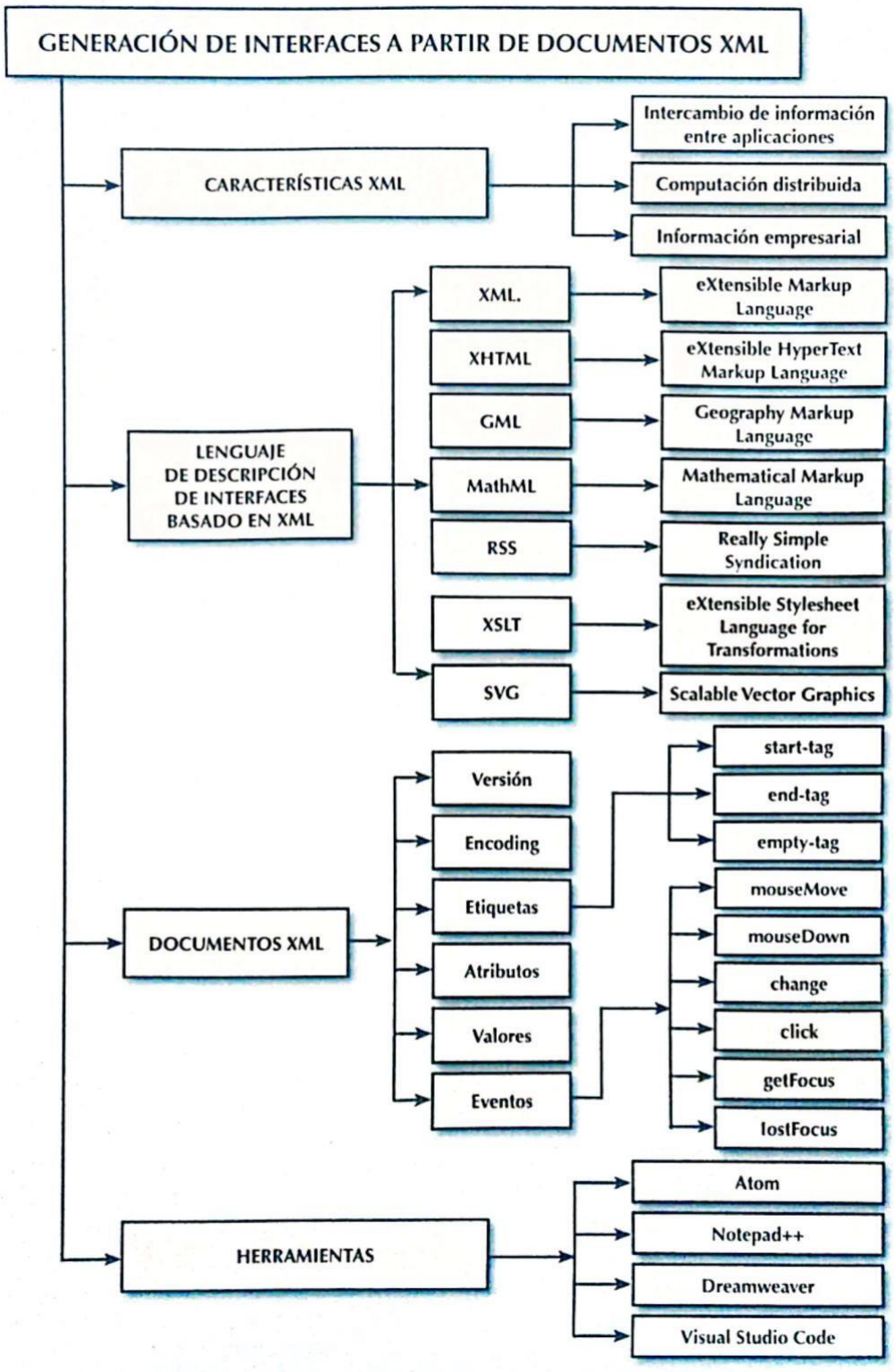
De una forma rápida y simple hemos visto como vincular imagenes en java.

UD2. Generación de interfaces de documentos XML

Objetivos

- Reconocer las ventajas de generar interfaces de usuario a partir de su descripción XML.
- Generar la descripción del interfaz en XML usando un editor gráfico.
- Analizar el documento XML generado.
- Modificar el documento XML.
- Generar el código correspondiente al interfaz a partir del documento XML.
- Programar una aplicación que incluya el interfaz generado.

Mapa conceptual



Glosario

Atributo. Componente de las etiquetas que consiste en un par nombre=valor.

Etiqueta. Es la marca que sirve para diferenciar un contenido específico del resto del contenido del documento. Una etiqueta empieza con el carácter “<”, continúa un nombre identificativo y termina con el carácter “>”.

GML. (Geography Markup Language). Lenguaje de marcado geográfico. Este tipo de documentos están compuestos de marcas precedidas de doble punto (:), que define el tipo de texto (títulos, párrafos, listas...).

MathML. (Mathematical Markup Language). Lenguaje de marcado matemático. Este lenguaje tiene el objetivo de intercambiar información entre programas de tipo matemático.

RSS (Really Simple Syndication). Sindicación realmente simple. Este lenguaje tiene como objetivo difundir información a usuarios que se han suscrito a una fuente de contenidos actualizada frecuentemente.

SVG (Scalable Vector Graphics). Gráficos vectoriales escalables. Este lenguaje permite representar elementos geométricos vectoriales, imágenes de mapa de bits y texto.

Valor. Es la asignación que se realiza al atributo. La estructura de un atributo XML es siempre un par de nombre-valor.

XHTML (eXtensible HyperText Markup Language). Lenguaje de marcado de hipertexto extensible. XHTML es un lenguaje derivado de XML similar a HTML, pero más aconsejable para la modelación de páginas web.

XML (eXtensible Markup Language). Lenguaje de marcado extensible. Es un lenguaje que es utilizado para estructurar, almacenar e intercambiar datos entre distintas plataformas.

XSLT (eXtensible Stylesheet Language for Transformations). Transformaciones de lenguaje de hoja de estilo extensible. Es el lenguaje de hoja de estilo recomendado para XML.

2.1. Lenguajes de descripción de interfaces basados en XML

El uso de la tecnología XML tiene un papel muy importante en la actualidad, ya que permite la compatibilidad entre sistemas para compartir información de manera fácil, segura y fiable. Se puede usar en bases de datos, editores de textos, hojas de cálculo y diferentes plataformas.

Este lenguaje en la actualidad es muy importante puesto que presenta diversos usos entre los que destacan:

- ***Intercambio de información entre aplicaciones***: al almacenar información mediante documentos de texto plano no se requiere software especial.

- ***Computación distribuida***: se trata de la posibilidad de utilizar XML para intercambiar información entre diferentes ordenadores a través de redes.

- ***Información empresarial***: este lenguaje tiene cada vez más importancia para generar interfaces empresariales gracias a la facilidad para estructurar los datos de la forma más apropiada para cada empresa.

Esta unidad tiene como propósito ser una guía para generar interfaces a partir de documentos XML, para lo que se estudiarán los lenguajes basados en XML y los principales editores de documentos XML.

XML, eXtensible Markup Language (lenguaje de marcado extensible) es un lenguaje utilizado para estructurar, almacenar e intercambiar datos entre distintas plataformas. Al ser un metalenguaje, puede ser empleado para definir otros lenguajes. Entre los más importantes actualmente se encuentran: XHTML, GML, MathML, RSS y SVG.

5.1.1. XHTML

eXtensible HyperText Markup Language (lenguaje de marcado de hipertexto extensible). XHTML es un lenguaje derivado de XML similar a HTML, pero con algunas diferencias que lo hacen más robusto y aconsejable para la modelación de páginas web. Los documentos XHTML tienen que cumplir:

- <! DOCTYPE> es obligatorio.
- El atributo xmlns en <html> es obligatorio.
- <html>, <head>, <title> y <body> son obligatorios.
- Los elementos siempre deben estar correctamente anidados.
- Los elementos siempre deben estar cerrados.
- Los elementos siempre deben estar en minúsculas.
- Los nombres de los atributos siempre deben estar en minúsculas.

- Los valores de los atributos siempre se deben citar.
- La minimización de atributos está prohibida.

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> ←es obligatorio
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<title>Un documento XHTML</title>  ←elemento cerrado
</head>
<body>
<h1>Cabecera principal del docunentoz<h1> ←elemento en minúsculas
<p>Nuestro primer párrafo</p>
<h2>Cabecera secundaria</h2>
<p>Otro párrafo con contenido distinto</p>
</body> ← Elementos
</html> ←  anidados
```

Figura 5.1 Código XHTML.

5.1.2. GML

Geography Markup Language (lenguaje de marcado geográfico). Un documento GML puede recibir un formato que define el tipo de texto que es (títulos, párrafos, listas...). Este tipo de documentos están compuestos de marcas precedidas de doble punto (:). En el siguiente ejemplo se muestra un listado de lenguajes definidos a partir del lenguaje XML.

```
:h1.Capítulo 5.- Generación de interfaces a partir de documentos XMLLenguaje XM
:p.Lenguajes de descripción de interfaces basados en XML
        :ol
        :li.GML
        :li.MathML
        :li.RSS
        :li.SVG
        :li.XHTML
        :eol.
```

Figura 2.2 Implementación código GML.

Actividad propuesta 2.1

Realiza un índice del capítulo anterior utilizando el lenguaje de marcado geográfico, GML.

5.1.3. MathML

Mathematical Markup Language (lenguaje de marcado matemático). Este lenguaje se usa junto con el lenguaje XHTML con el objetivo de intercambiar información entre programas de tipo matemático. En el siguiente ejemplo se muestra la fórmula matemática $a^2+b^2=c^2$ escrita en lenguaje MathML.

```
<math>
   <mrow>
        <mi>a</mi>
        <mn>2</mn>
        <mo>+</mo>
        <mi>b</mi>
        <mn>2</mn>
        <mo>=</mo>
        <mi>c</mi>
        <mn>2</mn>
   </mrow>
</math>
```

Figura 2.3 Implementación ejemplo MathML.

Actividad propuesta 2.2

Implementa la fórmula matemática a = Raíz cuadrada de $c^2 – b^2$ utilizando el lenguaje MathML.

5.1.4. RSS

Really Simple Syndication (sindicación realmente simple). Este lenguaje tiene como objetivo difundir información a usuarios que se han suscrito a una fuente de contenidos actualizada frecuentemente. Los programas de este tipo suelen estar compuestos por novedades del sitio web, el título, fecha de publicación o descripción. En el siguiente ejemplo se muestra una noticia publicada en lenguaje RSS.

```
<?xml version="1.0” encoding="UTF-8” ?>
 <rss version="2.0”>
    <channel>
          <title>Última hora</title>
          <description>Noticia importante
          </desciption>
          <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>
```

Figura 2.4 Implementación ejemplo RSS.

5.1.5. XSLT

eXtensible Stylesheet Language for Transformations (transformaciones de lenguaje de hoja de estilo extensible). Es el lenguaje de hoja de estilo recomendado para XML. XSLT es mucho más sofisticado que CSS. Con XSLT se pueden agregar o eliminar elementos y atributos desde o hacia el archivo de salida. También puede reorganizar y ordenar elementos, realizar pruebas y tomar decisiones sobre qué elementos ocultar y mostrar.

5.1.6. SVG

Scalable Vector Graphics (gráficos vectoriales escalables). Este lenguaje permite representar elementos geométricos vectoriales, imágenes de mapa de bits y texto. En los siguientes ejemplos se muestra la creación de diferentes elementos gráficos utilizando el lenguaje vectorial.

```
<!DOCTYPE html>
<html>
<body>
<svg height=""”100” width=”100”>
```

```
  <circle cx="50” cy="50” r=”"40” stroke="black”     stroke-width=”3” fill="darksalmon” />
  Lo siento, tu navegador no es compatible con SVG.
</svg>
</body> </html>
```

Figura 2.5 Ejemplo 1: implementación de código SVG círculo.

```
<svg width=”100” height=”100”»>
  <ellipse cx=”60” cy=”60” rx=”40” ry=”20” fill=”yellowgreen”/>
</svg>
```

Figura 2.6 Ejemplo 2: implementación de código SVG elipse.

Figura 5.7
Resultado del ejemplo 1.

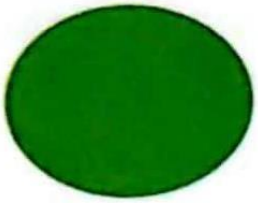


Figura 5.8
Resultado del ejemplo 2.

```
<svg width=”120” height=”120”>
   <rect x=”0” y=”0” width=""120” height="60” fill=""lightseagreen”/> </svg>
```

Figura 2.9 Ejemplo 3: implementación de código SVG rectángulo.

```
<svg width=""120” height=”120”»>
  <polygon fill=”SteelBlue” stroke="black” stroke-width=""2”
points=""15,90 45,30 75,90”/>
</svg>
```

Figura 2.10 Ejemplo 4: implementación de código SVG triángulo.

Figura 5.11 Resultado del ejemplo 3.

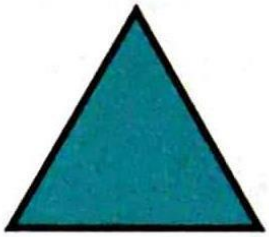


Figura 5.12 Resultado del ejemplo 4.

Actividad propuesta 2.3

Implementa la siguiente figura utilizando el lenguaje SVG.

Figura 5.13 Implementación ejemplo SVG.

5.2. El documento XML. Análisis y edición

Un archivo XML es un fichero en formato texto, que contiene etiquetas para identificar los elementos y datos que componen el documento. La primera línea del fichero debe ser:

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

- version: indica la versión de XML que se está empleando.
- encoding: especifica el juego de caracteres con el que se ha codificado el documento.

El resto del documento XML se escribirá con etiquetas, y siempre hay que abrirlas y cerrarlas. La estructura de las etiquetas de apertura y cierre siempre van a seguir un patrón como el que se muestra a continuación:

```
<etiqueta1> ..... </etiqueta1>
```

Es necesario que la estructura jerárquica se cumpla de manera estricta con respecto a las etiquetas que se utilizan en el documento. Es decir, todas 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 la que indica el cierre.

El conjunto formado por las etiquetas (apertura y finalización) y el contenido se conoce como elemento. Por ejemplo, en el caso de un conjunto del tipo <apellido> García </apellido> es un elemento o nodo, con la etiqueta "apellido" y el contenido "García". La clave de un documento bien estructurado es que para almacenar la información es necesario respetar este tipo de estructuras a la hora de establecer las etiquetas y los atributos.
Nota:
Un documento XML tiene una estructura anidada de manera jerárquica.
Es obligatorio cerrar todas las etiquetas, y si un elemento tiene vinculados otros elementos (hijos), elementos que descienden de él, se deben cerrar las etiquetas de los hijos antes de poder cerrar la etiqueta del padre.

5.2.1 Etiquetas

Las etiquetas XML son marcas que sirven para separar un contenido de otro en el documento. Una etiqueta empieza con el carácter "<", continúa un nombre identificativo, y termina con el carácter ">". El nombre de la etiqueta debe empezar por una letra, y continuar con 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 />

5.2.2 Atributos

Un atributo es un componente de las etiquetas estructurado de manera nombre=valor. Se puede encontrar en las etiquetas de apertura o en las etiquetas vacías, pero no en las de cierre. Hay que tener en cuenta que en una misma etiqueta no pueden existir dos atributos con el mismo nombre, siendo siempre todos los atributos de un elemento únicos. Por ejemplo:

```
<cliente nombre="Lucas" apellido1="Garcia-Miguel" apellido2="López" />
```

En este caso tenemos tres atributos únicos, nombre, apellido1 y apellido2. En cambio, en el siguiente caso no sería correcto, dado que tenemos el atributo apellido repetido:

```
<cliente nombre="Lucas" apellido="López" apellido="López">
```

5.2.3 Valores

El atributo de un elemento XML proporciona información acerca del elemento, es decir, sirven 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="Diseño de interfaces web" editorial= "Síntesis">
        <tipo>
            <libro isbn="9788491713241" edicion="1" paginas="210" />
        </tipo>
        <autor nombre="Diana Garcia-Miguel López" /> </texto>
</biblioteca>
```

Figura 5.14 Ejemplo de una estructura de biblioteca XML.

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

Actividad propuesta 5.4

Establece el formato que debería de tener una estructura XML para realizar una factura con los campos: factura, número, fecha, nombre cliente, teléfono cliente, dirección cliente, concepto y total.

5.2.4 Eventos

Los eventos proporcionan un mecanismo adecuado para tratar las diferentes formas de interacción entre el usuario y la aplicación, por ejemplo, cuando el usuario presiona una tecla o pulsa con el puntero del ratón. En algunas ocasiones, ante la llegada de un evento, nos

interesará tratarlo. En cambio, otras veces no será necesario el tratamiento del evento con ninguna acción.

El primer caso podría tratarse de la pulsación del botón siguiente en una interfaz en un proceso de instalación, el segundo caso en cambio puede ser rellenar los campos de un formulario.

Algunos de los eventos más comunes que se pueden producir en una aplicación como interacción con las interfaces gráficas son:

- MouseMove: evento producido al mover el ratón por encima de un control.
- MouseDown: este evento se produce al pulsar cualquier botón del ratón.
- Change: se produce al cambiar el contenido del control.
- Click: uno de los eventos más comunes se produce al hacer clic con el botón izquierdo del ratón sobre el control.
- GetFocus: este evento se activa cuando el control recibe el enfoque, es decir, cuando se activa el control en tiempo de ejecución para introducir datos o realizar alguna operación.
- LostFocus: es lo contrario del evento anterior, se activa cuando el control pierde el enfoque, es decir, se pasa a otro control para seguir introduciendo datos.

5.3. Herramientas libres y propietarias para la creación de interfaces de usuario multiplataforma

Una de las características fundamentales que presentan los editores de XML es la sencillez para escribir código resaltando la sintaxis, insertando elementos y estructuras de XML de uso común a través de la función de autocompletado. A continuación, se destacan algunas de las principales características de los editores más utilizados para el lenguaje XML.

	Atom	Notepad++	Dreamweaver	Visual Studio Code
Multinivel	✔	✔	✔	✔
Paneles	✔	✔	✔	✔
Control versiones	✔			✔

Libre	✔	✔	✔	✔
WYSIWYG			✔	✔

Cuabro 5.1 Comparativa de los editores

a) ***Atom***. El editor Atom es una herramienta multinivel que sirve tanto para las personas que comienzan a programar como para uso profesional. Además se pueden añadir lenguajes que no vienen de serie u otros tipos de interfaces gráficas.

El espacio de trabajo se compone de paneles que se pueden recolocar de manera flexible para que la programación resulte más cómoda. Además, tiene una manera de colaborar en tiempo real mediante la herramienta Teletype, permitiendo que muchos desarrolladores puedan editar un archivo a la vez en tiempo real. Por todos estos motivos, esta herramienta es muy útil cuando varios desarrolladores tienen que trabajar de forma colaborativa con otras personas.

Nota:

Una de las mayores ventajas del editor de código Atom son las herramientas Git y GitHub, que permiten controlar distintas versiones de un proyecto mientras se está desarrollando. Aunque Visual Studio Code ya lo incluye también en sus últimas versiones.

b) ***Notepad++.*** Este editor reconoce la sintaxis de múltiples lenguajes de programación. Es gratuito, está disponible para Linux y Windows y su código fuente se puede descargar.

Ha tenido un gran éxito entre los desarrolladores por las características que ofrece, lo ligero que es y su rapidez. Su interfaz puede ser personalizada y posee diversos plugins entre los que destaca el llamado XML Tools que añade un menú con opciones específicas como validar un documento XML con su DTD.

c) ***Adobe Dreamweaver CC***. Es un editor de código que permite escribir un documento mostrando directamente el resultado final. Por lo tanto, tú eliges si quieres programar con una presentación visual en tiempo real o la manera tradicional. Es compatible con Windows y OS X. Su principal fortaleza es que dispone de vista previa, de esta manera los desarrolladores pueden programar mientras previsualizan el producto final.

d) ***Visual Studio Code***. Es uno de los editores de código fuente más utilizados. Es compatible con varios lenguajes de programación y está disponible para Windows, Linux y macOS. Permite el resaltado de sintaxis, la finalización inteligente de código, tiene interfaz personalizable y es gratuito. Ofrece el servicio Live Share, extensión que permite compartir el código con otro programador, de forma que se puede colaborar en tiempo real.

5.3. Generación de código XML para datos en diferentes plataformas

Uno de los retos más importantes a los que se enfrentan los desarrolladores es el intercambio de datos entre sistemas incompatibles. El intercambio de datos con XML reduce en gran medida la dificultad ante la que se enfrentan los desarrolladores en cuanto al intercambio de datos entre sistemas incompatibles.

Gracias a su portabilidad, XML se ha convertido en 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. Esto es debido a que XML es un lenguaje independiente de la plataforma, lo que significa que cualquier programa diseñado para lenguaje XML puede leer y procesar los datos XML independientemente del hardware o del sistema operativo, La tecnología XML se basa en tres pilares fundamentales que permiten generar código para diferentes plataformas, estos pilares son el uso de XML para:

1. 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 desean recuperar.

2. 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 basado 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.

3. 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.

Actividad propuesta 5.5

Enumera tres casos prácticos en los que XML se utilice para representación de metadatos, como medio de intercambio de contenidos y como almacenamiento de contenidos.

La creación de tablas resulta clave en el desarrollo de interfaces, puesto que permite estructurar y organizar la información de forma eficaz, mejorando sustancialmente la experiencia de uso de los usuarios. Por ejemplo, en este caso práctico se ha diseñado una interfaz utilizando un documento XML que permite mostrar la información de manera estructurada. El código implementado quedaría como se muestra en la figura 5.15.

Cuabro 5.2

Interfaz resultado del código XLM de la figura 5.15

Datos aplicaciones	
Nombre app	% Uso
Instagram	30
Facebook	10
Tik Tok	25
Twitter	15
Linkedin	10

```xml
<?xml version="1.0" encoding="UTF-8"?>
<datos>
    <dato>
        <nombre>Instagram</nombre>
        <uso>30</uso>
    </dato>
    <dato>
        <nombre>Facebook</nombre>
        <uso>10</uso>
    </dato>
    <dato>
        <nombre>Tik Tok</nombre>
        <uso>25</uso>
    </dato>
    <dato>
        <nombre>Twitter</nombre>
        <uso>15</uso>
    </dato>
    <dato>
        <nombre>LinkedIn</nombre>
        <uso>10</uso>
    </dato>
</datos>
```

Figura 5.15 Datos implementados en XML.

5.4. Como leer archivos XML desde Java

Para leer y escribir archivos XML desde Java necesitaremos usar las librerías SAX o DOM, la primera solo permite la lectura de archivos XML, la librería DOM sin embargo nos permite tanto la lectura como la escritura de archivos XML desde Java.

Los pasos para leer un archivo XML desde Java serán los siguientes:

1. Crear un archivo XML de muestra

Utilizaremos un archivo XML de muestra students.xml como ejemplo. Contiene algunos datos sobre estudiantes y los leeremos utilizando código Java.

Crea un nuevo archivo llamado students.xml en el directorio ~/project con el siguiente contenido:

```
<students>
  <student id="101">
    <Name>John</Name>
    <id>11001</id>
    <location>India</location>
  </student>
  <student id="102">
    <Name>Alex</Name>
    <id>11002</id>
    <location>Russia</location>
  </student>
  <student id="103">
    <Name>Rohan</Name>
    <id>11003</id>
    <location>USA</location>
  </student>
</students>
```

2. Importar las bibliotecas necesarias

Utilizaremos las siguientes bibliotecas para leer un archivo XML utilizando código Java:

org.w3c.dom.*
javax.xml.parsers.*

Agrega las siguientes declaraciones al principio del archivo de código para importar las bibliotecas necesarias:

```
import org.w3c.dom.*; import org.xml.sax.SAXException;
import javax.xml.parsers.*; import java.io.*;
```

3. Analizar el archivo XML

Crea una nueva clase Java llamada Main en el directorio ~/project con el siguiente contenido:

```
public class Main {
  public static void main(String[] args) throws IOException, ParserConfigurationException,
SAXException {
      DocumentBuilderFactory dBfactory = DocumentBuilderFactory.newInstance();
      DocumentBuilder builder = dBfactory.newDocumentBuilder();
      // Fetch XML File
      Document document = builder.parse(new File("students.xml"));
document.getDocumentElement().normalize();
      //Get root node
      Element root = document.getDocumentElement();
      System.out.println(root.getNodeName());
      //Get all students
      NodeList nList = document.getElementsByTagName("student");
      System.out.println("..............................");
   }
}
```

Hemos creado una instancia del constructor y analizado el archivo XML utilizando el método parse. Después de eso, obtenemos el elemento raíz del documento y lo normalizamos y luego imprimimos su nombre. Después de eso, obtenemos a todos los estudiantes utilizando el método getElementsByTagName e imprimimos un separador.

4. Extraer datos de cada elemento

Para extraer datos de cada elemento, iteraremos a través de cada etiqueta del documento utilizando un bucle. Para cada estudiante, obtendremos sus detalles como ID, Nombre, Número de Matrícula y Ubicación.

Agrega el siguiente código dentro del bucle:

```
Node node = nList.item(i);
System.out.println();    //Imprime una línea en blanco como separador if
(node.getNodeType() == Node.ELEMENT_NODE) {
//Print each student's detail
    Element element = (Element) node;
    System.out.println("Student id : " + element.getAttribute("id"));
    System.out.println("Name : " +
    element.getElementsByTagName("Name").item(0).getTextContent());
    System.out.println("Roll No : " +
    element.getElementsByTagName("id").item(0).getTextContent());
    System.out.println("Location : " +
    element.getElementsByTagName("location").item(0).getTextContent());
}
```

El código anterior extraerá los datos de cada elemento del archivo XML. Utilizando el método getAttribute, se recupera el ID de cada estudiante. Utilizando los métodos getElementsByTagName y getTextContent, se recuperan el Nombre, el Número de Matrícula y la Ubicación de cada estudiante.

Ejecutar el código Java

Compila y ejecuta el código en la terminal:

javac Main.java && java Main Deberías

ver la siguiente salida:

students

.....

Student id : 101

Name : John

Roll No : 11001

Location : India

Student id : 102

Name : Alex

Roll No : 11002

Location : Russia

Student id : 103

Name : Rohan

Roll No : 11003

Location : USA

5.5. Como leer archivos HTML/XHTML desde Java y asignarlos a un JTextPanel

Para poder insertar en un componente JtextPane un archivo HTML/XHTML y presentar el resultado en pantalla como si fuera un navegador, es decir interpretando el código HTML/XHTML primero deberemos cargar o leer el archivo HTML/XHML desde Java y luego interpretarlo a través de un componente visual como JTextPanel.

5.1. Lectura archivos y carga en un String

Para poder hacer la lectura del archivo HTML/XHTML y dejarlo en una variable tipo String, usaremos el siguiente código:

```
InputStream fop = Main.class.getResourceAsStream("file.txt");
String line; try {
        BufferedReader reader = new BufferedReader(new InputStreamReader(fop, "UTF-8"));
        while ((line = reader.readLine()) != null) {
        /* Aquí agregamos nuestra lógica */
        }
} finally {
        fop.close();
}
```

La primera línea localiza el archivo, por así decirlo, en el mismo directorio que nuestra clase **Main** se encuentra.

La siguiente línea importante crea un BufferedReader, y establece la codificación de nuestro archivo para efectuar la lectura.

```
BufferedReader reader = new BufferedReader(new InputStreamReader(fop, "UTF-8"));
```

Por último, realizamos una lectura secuencial del archivo usando

```
while ((line = reader.readLine()) != null)
```

Y tendremos en **line** una cadena de caracteres que podremos modificar como queramos o cargarla en un elemento de nuestra interfaz. Espero que os sea útil.

5.2. Interpretar un String que contiene código HTML con JTextPanel

Para poder interpretar código HTML/XHTML contenido en un String usando un componente Jtext Panel haremos lo siguiente:

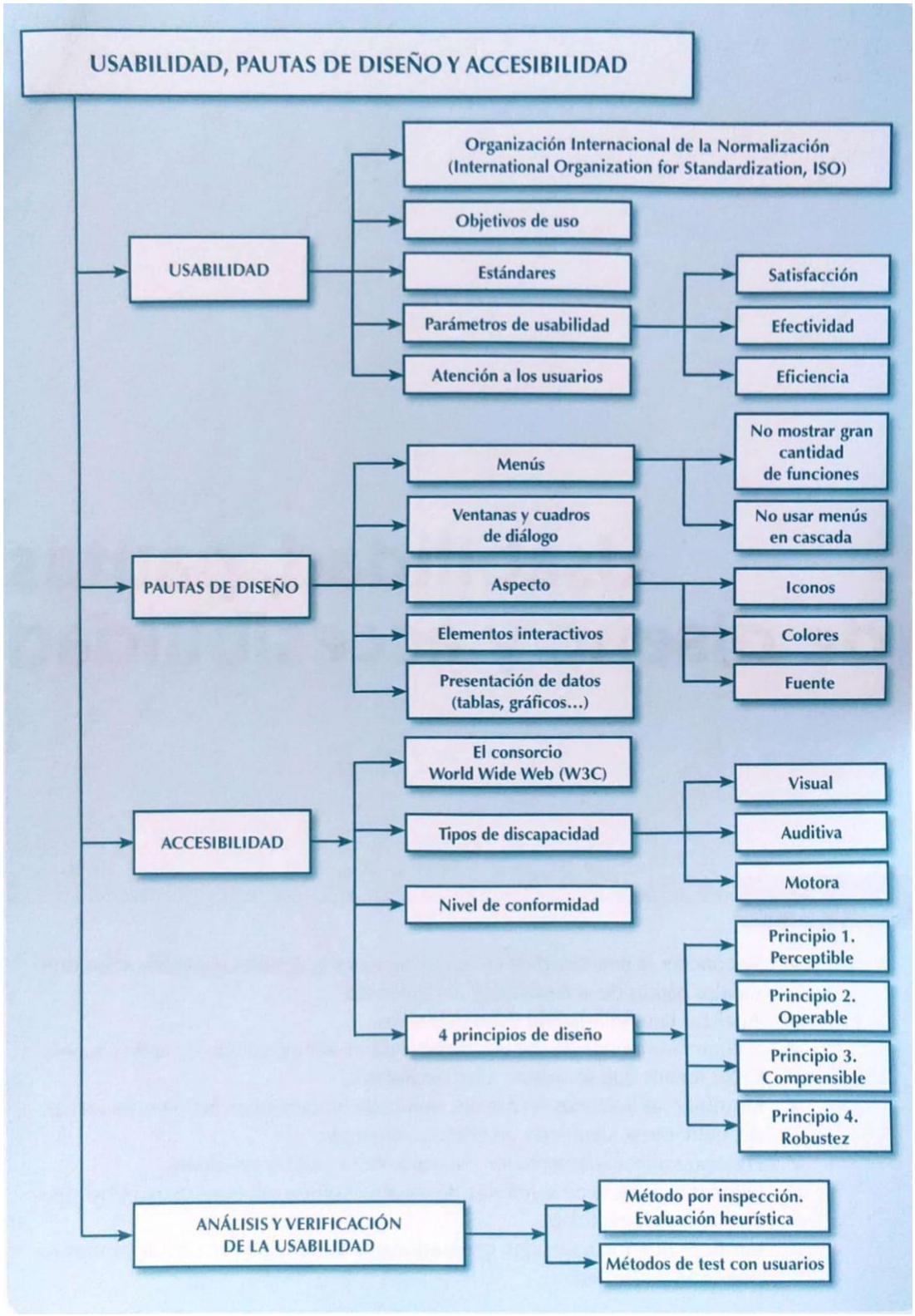
```
JTextPanel panelTexto = new JTextPanel(); panelTexto.setContentType("text/html"); panelTexto.setText(line);
//line, contiene el código HTML a interpretar.
```

UD4. Usabilidad y accesibilidad

Objetivos

- Reconocer la necesidad de crear aplicaciones accesibles e identificar las principales pautas de accesibilidad al contenido.
- Analizar la usabilidad de diferentes sitios.
- Valorar la importancia del uso de estándares en la creación de aplicaciones.
- Crear menús que se ajusten a los estándares.
- Distribuir las acciones en menús, barras de herramientas, botones de comando, entre otros, siguiendo un criterio coherente.
- Distribuir adecuadamente los controles en la interfaz de usuario.
- Diseñar el aspecto de la interfaz de usuario (colores y fuentes entre otros) atendiendo a su legibilidad.
- Verificar que los mensajes generados por la aplicación son adecuados en extensión y claridad.

Mapa conceptual


USABILIDAD, PAUTAS DE DISEÑO Y ACCESIBILIDAD
USABILIDAD
Organización Internacional de la Normalización (International Organization for Standardization, ISO)
Objetivos de uso
Estándares
Parámetros de usabilidad
Atención a los usuarios
Satisfacción
Efectividad
Eficiencia
PAUTAS DE DISEÑO
Menús
Ventanas y cuadros de diálogo
Aspecto
Elementos interactivos
Presentación de datos (tablas, gráficos...)
No mostrar gran cantidad de funciones
No usar menús en cascada
Iconos
Colores
Fuente
ACCESIBILIDAD
El consorcio World Wide Web (W3C)
Tipos de discapacidad
Nivel de conformidad
4 principios de diseño
Visual
Auditiva
Motora
Principio 1. Perceptible
Principio 2. Operable
Principio 3. Comprensible
Principio 4. Robustez
ANÁLISIS Y VERIFICACIÓN DE LA USABILIDAD
Método por inspección. Evaluación heurística
Métodos de test con usuarios

Glosario

Consistencia. Una aplicación se puede calificar como consistente cuando el usuario puede entender y asimilar las funcionalidades de la misma sin necesidad de recibir instrucciones.

Discapacidad auditiva. La discapacidad auditiva se define como el déficit total o parcial en la percepción del sonido, que se evalúa por el grado de pérdida de la audición en cada oído. Se pueden distinguir entre las personas hipoacúsicas y personas sordas

Discapacidad motora. Una persona con una discapacidad motora es aquella que sufre de una manera duradera y frecuentemente crónica una afección más o menos grave del aparato locomotor que supone una limitación de sus actividades.

Discapacidad visual. e define así la dificultad que presentan algunas personas para par-

ticipar en actividades propias de la vida cotidiana, como consecuencia de una disminución o pérdida de las funciones visuales y las barreras presentes en el contexto en que desenvuelve la persona.

Heurístico (análisis). Búsqueda e identificación de errores para la mejora y optimización del producto para asegurar la usabilidad del mismo.

ISO (International Organization for Standardization). La Organización Internacional de la Normalización 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.

W3C. El consorcio World Wide Web es una comunidad internacional donde las organizaciones miembros se encargan del desarrollo de estándares que aseguran el crecimiento y el acceso a la web, cuyo objetivo principal era posibilitar el desarrollo de tecnologías interoperables.

WAI (Web Accessibility Initiative). Iniciativa para la accesibilidad a la web, que se crea en el marco del W3C.

4.1. Usabilidad

En plena era digital, el número de contenidos proporcionados a través de aplicaciones web que se muestran a los usuarios a través de una interfaz, se ha visto aumentado de manera exponencial.
Además, el usuario cada vez cuenta con más formación relativa al uso de las nuevas tecnologías. Es por ello que las aplicaciones más recientes tienen una apariencia y unas funcionalidades que costaba pensar hace apenas unos años.

"Si no lo haces fácil, los usuarios se marcharán de tu web", Jakob Nielsen.

Por ello, para el éxito de una aplicación o aplicación web, es cada vez más importante proporcionar un acceso rápido a los contenidos y proporcionar una experiencia visual agradable al usuario, que lo invite a volver en un futuro para obtener la información que necesita. La consecución de los fines de una aplicación dependerá en gran medida de la satisfacción que se proporcione al usuario. Dicha satisfacción dependerá, a su vez, de una serie de parámetros que guardan relación con la claridad y utilidad de los contenidos, con la calidad de los mismos, con un diseño atractivo, etc. Por tanto, es fundamental un diseño adecuado de la web para facilitar el intercambio de información con el usuario.

Actividad propuesta 4.1
¿Qué parámetros consideras que se deben tener en cuenta para conseguir la usabilidad deseada? ¿Parámetros puramente estéticos o también relaciones con el contenido, o con la estructura con la que los elementos de la aplicación han sido distribuidos?

4.1.1. Objetivos de uso y estándares de usabilidad

De acuerdo con Nilsen, la usabilidad es un concepto relacionado intrínsecamente con la forma en la que una interfaz es presentada al usuario, así como la forma en la que el usuario la utiliza, teniendo en cuenta algunos parámetros como su sencillez, claridad, etc.

La Organización Internacional de 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. Según la ISO, la usabilidad hace mención a la capacidad de un software determinado para ser comprendido, utilizado y aprendido por el usuario, al mismo tiempo que le resulta atractivo.

Una aplicación no solo tiene que tener un aspecto atractivo y ser tecnológicamente puntera, sino que además debe ser fácilmente comprensible por cualquier usuario, con el fin de proporcionar la información que este busca en el menor tiempo. Por ello, se puede afirmar que la usabilidad depende del producto, pero también depende del usuario, en cuanto a cómo interactúa con la interfaz de la aplicación en concreto y cómo la aprecia en cuanto a sencillez y facilidad de utilización.

Esto se pone de manifiesto a través de múltiples hechos que ocurren diariamente en la relación usuario-interfaz. Por ejemplo, cuando por la causa que fuere la experiencia de navegación a través de un determinado portal no es agradable, o bien la información que proporciona no es clara o útil, es muy probable que el usuario abandone la aplicación y no vuelva en un futuro.

En base a lo anterior, podemos afirmar que existen parámetros subjetivos (satisfacción de usuario) y objetivos (tiempo empleado por el usuario para conseguir su objetivo, errores cometidos para conseguir lo que se busca, etc.) para poder medir la usabilidad de un determinado sitio.

Para conseguir una interfaz con buena usabilidad, antes de comenzar un proyecto determinado, es importante tener en cuenta algunas cuestiones como las siguientes:

1. ¿Qué se le está ofreciendo al usuario?

2. ¿Quiénes son los potenciales usuarios y qué formación o conocimientos tendrán?

3. ¿Qué necesitarán o buscarán los usuarios?

4. ¿En qué contexto se moverán los potenciales usuarios?

Figura 4.1 Principales normas sobre usabilidad.

Existen diversos estándares y normas relacionados directamente con la usabilidad y con la accesibilidad, que definen diferentes aspectos relativos a esta. Se intenta conseguir así una uniformidad en los criterios de diseño, puesto que no sería lógico que cada diseñador de interfaces escogiera unos parámetros de usabilidad distintos:

- ISO / IEC 9126. Se trata de un estándar internacional para la evaluación de la calidad del software, se presenta dividido en cuatro partes: modelo de la calidad, métricas externas, métricas internas y métricas de calidad en uso.

- ISO / DIS 9241-11. Se trata de una norma que recoge los beneficios que aporta la medida de la usabilidad en términos de resultados y satisfacción obtenidos por el usuario. Estos beneficios se miden por el grado de consecución de los objetivos previstos en cuanto a utilización, por los recursos empleados para alcanzar estos objetivos y por el grado de aceptación del producto por parte del usuario.

- ISO 13407. La ISO 13407 proporciona una guía para alcanzar la calidad en el uso mediante la incorporación de actividades de naturaleza iterativa involucradas en el diseño centrado en el usuario (DCU).

- ISO 9241 / 151. Ergonomía de la interacción hombre-sistema. Parte 151: Directrices para las interfaces de usuario web. Proporciona directrices sobre el diseño centrado en el usuario para las interfaces de usuario web con el objetivo de aumentar su usabilidad.

- UNE 139803:2004. Bajo el título de “Aplicaciones informáticas para personas con discapacidad. Requisitos de accesibilidad para contenidos en la web”, es una norma española, publicada en diciembre de 2004, que contempla las especificaciones que han de cumplir los contenidos web para que puedan ser accesibles. Se trata de una transposición de las “Pautas de accesibilidad al contenido en la web” (WCAG 1.0) desarrolladas por la iniciativa WAI de W3C, pero estructuradas de forma diferente.

- UNE 139803:2012. Dadas las diferencias entre las WCAG 2.0 y las WCAG 1.0 surgió la necesidad de actualizar el contenido de esta norma UNE para que sus requisitos sean acordes con el contenido de las WCAG 2.0. Así, en 2012 se actualizó esta norma UNE para adoptar directamente las WCAG 2.0. Esta norma UNE señala directamente qué partes de WCAG 2.0 se consideran requisitos y con qué nivel de prioridad.

Es imprescindible tener en cuenta las denominadas medidas de usabilidad, que son una herramienta clave que permite evaluar la usabilidad en cuanto al desarrollo de interfaces se refiere.

Actividad propuesta 4.2

¿Cuál consideras que es el objetivo principal de la interacción persona-ordenador (IPO), o en inglés Human Computer Interaction (HIC), la disciplina que estudia el intercambio de información entre las personas y los ordenadores?

Una de las herramientas más utilizadas son los test de usabilidad, los cuales se encargan de evaluar desde la facilidad de uso de una aplicación por parte de un usuario hasta si la funcionalidad implementada 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, no estará cumpliendo los criterios de usabilidad. Los test de usabilidad se han de desarrollar de forma exhaustiva para que de manera objetiva se evalúen todos los parámetros deseados. Estos test han de contemplar ciertas métricas que se exponen a continuación, reactivas a tres importantes parámetros: satisfacción, efectividad y eficiencia.

Satisfacción: el nivel de satisfacción de un usuario es clave para la evaluación de la aplicación. Las métricas que se contemplan bajo este parámetro son: calificación de satisfacción del usuario sobre la aplicación, frecuencia de reutilización de la aplicación, calificación relativa a la facilidad de aprendizaje o la medida de uso voluntario de la aplicación.

Efectividad: determina el grado de éxito de una aplicación. Este parámetro está estrechamente ligado también con la facilidad de aprendizaje de la herramienta. Se deben tener en cuenta las siguientes métricas para su evaluación: cantidad de tareas relevantes completadas en cada uno de los intentos, número de accesos a la documentación, al soporte y a la ayuda, cantidad de funciones aprendidas o cantidad y tipos de errores tolerados por los usuarios, entre otras.

Eficiencia: se define de manera relativa al tiempo que se requiere para completar una determinada tarea con el software implementado. Las métricas en torno a este atributo se basan en el primero de los intentos: tiempo productivo de uso, tiempo para aprender el funcionamiento de la interfaz, eficiencia relativa al primer intento o errores persistentes, entre otros.

4.1.2. Los usuarios

Como hemos visto, la usabilidad se centra en solucionar y dar respuesta a las posibles casuísticas que debe presentar una interfaz para poder ofrecer una grata experiencia de navegación, sea cual fuere el usuario que la utilice. Se debe dar, por tanto, respuesta a gran diversidad de capacidades. Algunas de las características más habituales que se han de tener en cuenta son:

- Capacidades cognitivas y perceptivas. Comprensión del lenguaje, capacidad de aprendizaje y asimilación de conceptos, resolución de problemas.

- Culturales. Diversidad lingiística o nivel cultural. Esto puede afectar en la interpretación de formatos, medidas, títulos sociales, signos de puntuación, protocolos y formalidades.

- Discapacidades. Una de las casuísticas más importantes que tener en cuenta durante el proceso de desarrollo de un sitio web, en cuanto a la usabilidad y en concreto a la accesibilidad, es la adecuación del programa desarrollado a las personas con algún tipo de discapacidad.

- Tecnológica. Conexión a Internet, tamaños de pantalla, requisitos de memoria y proceso.

Para finalizar este apartado, es interesante considerara la siguiente clasificación, en función del tipo de usuario que puede utilizar una determinada aplicación, lo que permite definir ciertos parámetros como los permisos de acceso de una persona o los recursos a los que puede o no acceder.

Cuadro 4.1 Tipos de usuarios en función de sus permisos de acceso o su finalidad

Usuario anónimo. Un usuario anónimo es aquel que navega por la página sin identificarse como usuario registrado o sin tener sesión creada, Por ejemplo, en el caso de un banco, un usuario sin registrar podrá acceder a la página de inicio, normalmente con información básica de contenido, pero no tendrá acceso a la zona privada.

Usuario final registrado. Cuando se implementa esta opción, normalmente se hace para disponer de ciertos privilegios o recordar datos de sesión para agilizar el proceso de navegación.

Usuario beta tester. En el proceso del desarrollo de software se suele crear un perfil para un usuario usado como tester, cuyo fin es realizar las operaciones oportunas para verificar que la aplicación funciona según los requisitos del cliente. Reportan los fallos encontrados a los diseñadores de la aplicación que se encargan de solucionarlos antes de implantarlo definitivamente.

4.2. Pautas de diseño. Estructura de la interfaz de una aplicación

El diseño de la estructura de una interfaz no es un hecho trivial que deba dejarse al azar, sino que se va a tener en cuenta un conjunto de pautas de diseño que garanticen un mejor resultado final, por ejemplo, la ubicación de las ventanas, el diseño de cuadros de mensaje y de ventanas de diálogo, imágenes o iconos, entre otras. A continuación, se exponen las diferentes pautas que se deben abordar para el diseño de una interfaz:

4.2.1. Pautas de diseño para menús

Para la implementación de un buen diseño, todo menú va a permitir una adecuada navegación por la aplicación mostrando todas las condiciones y permitiendo al usuario seleccionar las acciones mostradas en este menú, siempre y cuando se adecuen a los permisos del usuario en dicha aplicación. Siempre se debe indicar el título del menú, y cuando este se muestra al usuario debe contener las opciones y la acción asociada a cada una de ellas, con el objetivo de facilitar la selección de la opción que más se ajuste a sus necesidades.

Otra de las pautas de diseño más importantes es definir una zona concreta en la que se mostrarán las diferentes opciones y que no variará a lo largo del diseño, ya que, de lo contrario, el usuario deberá buscar en cada nueva ventana donde ha sido colocado el menú disminuyendo la satisfacción de uso sobre la aplicación.

Habitualmente se coloca en la parte superior de las aplicaciones, pero esta dependerá del diseño implementado. Los menús “generales” que se sitúan de forma estática en la interfaz de una aplicación suelen desplegarse en forma de cascada, mostrando nuevos submenús cuando se accede a ellos. Además de este tipo de menús, también es posible encontrar los denominados “contextuales” o “emergentes”, que aparecen al seleccionar un objeto concreto.

En este tipo de casos se deben tener en cuenta las siguientes pautas de diseño:

1. No usar menús en cascada.

2. No mostrar una cantidad excesiva de funciones en este tipo de menús. Es aconsejable reducir las opciones a un máximo de diez elementos.

3. Las funciones que se muestran en este tipo de menús también deben estar contenidas en otro sitio, habitualmente en el menú “general” que aparece fijo en la aplicación.

Como crear una barra de menú con JMenuBar

JMenuBar es un componente de Java Swing que se utiliza para crear una barra de menú en un JFrame. Una barra de menú puede contener varios JMenu y cada JMenu puede contener varios JMenuItem. Los menús son desplegables y permiten al usuario ejecutar diferentes acciones en la aplicación.

Este ejemplo muestra cómo crear una barra de menú con opciones de menú y añadirla a un JFrame.

```
import javax.swing.*;

public class EjemploJMenuBar £
public static void main(String[] args) 4
// Crear un marco
JFrame frame = new JFrame("Ejemplo de JMenuBar");
frame.setSize(400, 300); frame
.setDefaultClose0peration(JFrame.EXIT_ON_CLOSE);

// Crear la barra de menú
JMenuBar menuBar = new JMenuBar();

// Crear un menú
JMenu menuArchivo = new JMenu("Archivo");

// Crear items del menú

JMenuItem itemAbrir = new JMenuItem("Abrir");
JMenuItem itemGuardar = new JMenuItem("Guardar");
```

```
JMenuItem itemSalir = new JMenuItem("Salir");

// Añadir los items al menú menuArchivo.add(itemAbrir);
menuArchivo.add(itemGuardar);
menuArchivo.addSeparator(); // Añadir un separador
menuArchivo.addCitemSalir);

// Añadir el menú a la barra de menú menuBar
.add(menuArchivo);

// Establecer la barra de menú en el marco frame.
setJMenuBar(menuBar);

frame.setVisible(true);
```

4.2.2. Pautas de diseño para ventanas y cuadros de diálogo

El diseño y creación de ventanas es uno de los elementos clave en el desarrollo de una aplicación, por esta razón, su diseño y posterior implementación debe realizarse teniendo en cuenta algunos aspectos muy importantes, como que los usuarios sean capaces de abrir y cerrar ventanas para que estas no se interpongan impidiendo visualizar lo que aparece en la pantalla a la que desean acceder, también se deben habilitar los mecanismos oportunos para que los usuarios puedan modificar el tamaño de las ventanas, entre otras. El número de ventanas debe escogerse con especial atención, no es aconsejable utilizar un gran número de ellas que puedan saturar la pantalla y, por lo tanto, al usuario. Pero tampoco debe reducirse en exceso el número de ventanas, puesto que las que sean utilizadas quedarán saturadas de contenido.

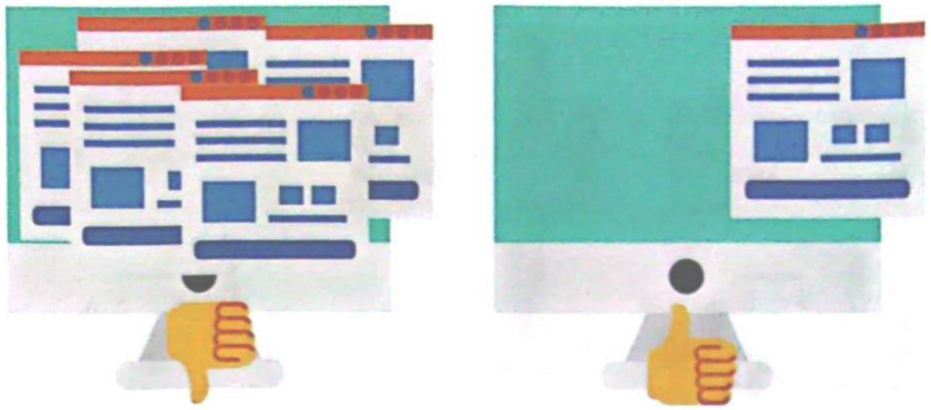


Figura 4.3 Interfaz con múltiples ventanas e interfaz simplificada.

Los cuadros de diálogo, al igual que las ventanas, son unos de los elementos más importantes y que permiten definir y adecuar el diseño de una aplicación a las necesidades de los futuros usuarios. Los cuadros de diálogo son los que permiten establecer una comunicación activa entre los usuarios y la interfaz. A través de cajas de texto, habitualmente de tipo emergente, se envían mensajes al usuario, que deberá actuar en consecuencia a estos. Los mensajes deben ser activos y positivos, indicando toda información relevante que deba conocer el usuario y sin dar por sentado ningún tipo de información.

Figura 4.4 Cuadro de diálogo claro y diálogos sin posibilidad de solución.

Crear cuadros de diálogo con JDialog

Un JDialog es el componente Java para hacer ventanas secundarias en una interface de usuario con javax.swing. Está entre JFrame, que sería la ventana principal y JOptionPane, que serían ventanas secundarias para mostrar notificaciones, pedir confirmaciones o pedir algún dato rápido al usuario. Un JDialog es más adecuado para presentar una ventana secundaria con más contenido que un JOptionPane.

Ejemplo de JDialog modal

A veces las ventanas de *JOptionPane* no son suficientes para lo que necesitamos. Por ejemplo, podemos querer pedir un dato más complejo que una simple selección de una opción. Y a veces mientras el usuario introduce ese dato complejo en la ventana y pulsa "Aceptar", queremos que nuestro código se pare en espera de la respuesta. Para estas situaciones tenemos los *JDialog* modales. En un *JDialog* podemos poner todos los componentes que queramos, haciendo la ventana todo lo compleja que queramos. Si ese *JDialog* es además modal, en el momento de hacerla visible llamando a *setVisible(true)*, el código se quedará parado en esa llamada hasta que la ventana se cierre. Podemos, por tanto, justo después del *setVisible(true)*, pedirle a la ventana los datos que ha introducido el operador.

Vamos a hacer aquí un ejemplo para ver todo esto. Como no quiero complicar demasiado el *JDialog* por claridad del ejemplo, el *JDialog* sólo tendrá un *JTextField* dentro de él y al pulsar *<intro>* en ese *JTextField*, después de haber escrito algo, el *JDialog* se cerrará. El código para conseguir esto es el siguiente:

```java
package com.chuidiang.ejemplos.option_pane_dialog_modal;

import java.awt.Frame;
import java.awt.event.ActionEvent; import
java.awt.event.ActionListener;

import javax.swing.JDialog; import
javax.swing.JTextField;

/**
*     Dialogo modal que sirve de ventana de captura de datos.<br>
*     Contiene un JTextField en el que escribimos un texto y pulsando enter
después
*     de escribir en el, la ventana se cierra.
 *
*     @author Chuidiang
*      */
public class DialogoModal extends JDialog {
private JTextField textField;

    /**
*     Constructor que pone titulo al dialogo, construye la ventana y la hace
*     modal.
     *
*     @param padre
*     Frame que hace de padre de esta dialogo.
     */
    public DialogoModal(Frame padre) {

        // padre y modal
super(padre, true);
```

```
        setTitle("Mete un dato");
textField = new JTextField(20);
getContentPane().add(textField);

        // Se oculta la ventana al pulsar <enter> sobre el
textfield         textField.addActionListener(new ActionListener()
{
            @Override
            public void actionPerformed(ActionEvent arg0) {
setVisible(false);
            }
        });
    }

    /**
* Deveulve el texto en el jtextfield
     *
* @return el texto
     */
    public String getText() {
return textField.getText();
    }
}
```

Le hemos añadido además un método *getText()* que devuelve el texto del *JTextField*. Será este al método al que llamemos para conseguir el texto introducido por el usuario después de que se cierre la ventana.

Para hacer la llamada a este diálogo y recoger los resultados, usaremos el siguiente código:

```
DialogoModal dialogoModal = new DialogoModal((Frame) ventanaPadre);
dialogoModal.pack();  // para darle tamaño automático a la ventana.
dialogoModal.setVisible(true);

// Al ser modal, la llamada a setVisible(true) se queda bloqueada hasta // que
el dialogo modal se oculte. Por ello, a continuación tenemos
// la seguridad de que el texto ya esta disponible.
System.out.println("El usuario ha escrito "+dialogoModal.getText());
```

7.2.3. Pautas de diseño relativas al aspecto

El diseño relativo al aspecto de la interfaz de usuario pone el foco en los elementos esenciales: color, fuente y distribución de los elementos. El diseño de los elementos visuales debe facilitar y mejorar la usabilidad de la aplicación, una mala selección puede llevar al fracaso a la mejor de las aplicaciones.

a) Iconos: este tipo de elementos permite asociar un objeto a una acción concreta, dinamizando así la interacción entre la interfaz de la aplicación y el usuario, por lo tanto, la premisa principal en cuanto al diseño es que debe ser representativa de la acción con la que se vincula, siendo preferible escoger diseños simples y no excesivamente complejos.

Figura 4.5 Ejemplos de psicología Y del color aplicados al diseño de interfaces.

b) Colores: como se analizó al inicio de este libro, la elección de los colores resulta decisiva en la experiencia de los usuarios, puesto que, entre otras características, permite poner el foco de atención en aquellos elementos y funciones más importantes. Además, aportan identidad de la marca a la que se vincula la aplicación, por ejemplo, la interfaz de la aplicación para la entidad BBVA presenta los colores característicos de la marca.

c) Fuente: la tipografía es el tipo de letra utilizado para el diseño de una interfaz concreta. Es conveniente usar una de manera uniforme a lo largo de todo el diseño o, al menos, que para los mismos fines se utilice la misma tipografía. Debe escogerse un tipo de letra que facilite la lectura a los usuarios y, por tanto, mejore su experiencia de uso. Pero además de la tipografía se deben tener en cuenta: el tamaño de la fuente (que ha de ser el adecuado para la lectura) y el tipo de pantalla y dispositivo donde se va a utilizar la aplicación, el color y, finalmente, el estilo.

Figura 7.6 Interfaces con buena y mala tipografía.

4.2.4. Pautas de diseño para elementos interactivos

Los elementos interactivos aportan al diseño de cualquier interfaz cierto comportamiento dinámico que permite establecer una comunicación activa entre la interfaz de la aplicación y los usuarios de esta. Este tipo de elementos son analizados ampliamente en los capítulos iniciales de este libro: botones, checkBox, menús desplegables de selección o radioButton, entre otros.

Las pautas de diseño relativas a la inclusión de este tipo de componentes se deben dirigir hacia la mejora de la legibilidad de los mismos, por ejemplo, en las cajas de texto es conveniente añadir texto explicativo que ayude al usuario a conocer qué tipo de datos se deben utilizar para completar las cajas de texto, así mismo, también se aconseja que el tamaño de las cajas de texto se ajuste lo máximo posible a la ventana en la que son mostrados.

En cuanto a los botones, checkBox o radioButton, es decir, elementos que permiten seleccionar uno o varios valores y enviarlos a la aplicación para realizar las acciones oportunas, deben cumplir algunas pautas de diseño:

1. Los títulos deben ser intuitivos.

2. Las acciones codificadas en cada opción deben quedar lo suficientemente comprensibles para el usuario.

3. Las opciones deben ser fácilmente distinguibles unas de otras y, por tanto, relativamente rápidas de escoger y seleccionar.

7.2.5. Pautas de diseño para la presentación de datos

Una aplicación debe presentar diferentes conjuntos de datos y, además, la exposición de estos se hará de múltiples formas, por lo que el modelado de esta representación debe cumplir ciertas pautas de diseño en función del tipo de distribución escogida: tablas, gráficos...

En el caso de las tablas, la información se debe mostrar de forma estructurada, puesto que resultan clave para poner énfasis sobre un conjunto de datos para que el usuario les preste especial atención, ahora bien, no es conveniente abusar del uso de estos elementos, puesto que si se trivializa su uso el usuario no les prestará atención en el futuro. Algunas de las pautas de diseño principales relativas a las tablas son:

1. Utilizar etiquetas en filas y columnas, claras y concisas.

2. Incluir un título de la tabla cuya longitud sea inferior a dos líneas de texto.

3. Usar encabezados de fila o columna para resumir el contenido de la fila o columna.

Figura 4.7 Interfaz con excesivo uso de gráficos vs interfaz sencilla.

Finalmente, también será posible utilizar gráficos, pero al igual que en el caso de las tablas, el uso de este tipo de elementos no debe ser abusivo. Algunas de las pautas de diseño son:

1. Adecuar el tamaño de los gráficos a las dimensiones de la pantalla.

2. No abusar del número de gráficos.

3. Seleccionar una paleta de color que permita diferenciar los datos, además, es aconsejable utilizar una leyenda fácil de identificar en el gráfico.

4.3. Accesibilidad

El concepto de accesibilidad está relacionado de una manera muy directa con el de usabilidad. En este caso, la accesibilidad se puede definir como la posibilidad de acceso a una determinada aplicación, frente a la usabilidad que se refiere a la facilidad de uso. Por tanto, es evidente que una aplicación debe ser accesible antes que usable.

El acceso a una determinada aplicación debe facilitarse para todos los usuarios potenciales, más allá de las limitaciones técnicas de cada usuario (software, hardware, etc.) o de las limitaciones individuales de cada uno (discapacidades, dominio de un determinado idioma, etc.). De esta forma, una aplicación accesible debe tener en cuenta la gran diversidad de potenciales usuarios que puede llegar a tener.

Aunque la accesibilidad se encuentra especialmente dirigida hacia el desarrollo de aplicaciones web, es conveniente conocer algunos de sus conceptos y normas más importantes, ya que en muchos casos el diseño de una aplicación va a ser extrapolable a diferentes ámbitos.

4.3.1. El consorcio World Wide Web (W3C)

El consorcio World Wide Web (W3C) es una comunidad internacional donde las organizaciones miembro se encargan del desarrollo de estándares que aseguran el crecimiento y el acceso a la web. Fue creada en 1994 con un conjunto de objetivos que permitieran desarrollar tecnologías interoperables.

Dentro de este marco se hace necesario desarrollar estrategias, directrices y recursos para garantizar el acceso por igual a la web, es así como aparece la Web Accessibility Initiative o Iniciativa para la Accesibilidad a la Web (WAI). Esta iniciativa desarrolló las Directrices de Accesibilidad para el Contenido Web 2.0, más conocido como WCAG 2.0, donde se recogen pautas y técnicas que permitan ofrecer soluciones accesibles para el software y contenido web. Este conjunto de pautas fue aprobado bajo el estándar internacional ISO/IEC 40500:2012. En junio de 2018 fue sustituido por la versión WCAG 2.1.

Los cuatros principios que regulan este funcionamiento son que el diseño debe ser perceptible, operable, comprensible y robusto.

Tal y como se recoge en la Recomendación del W3C del 11 de diciembre de 2008 para Pautas de Accesibilidad para el Contenido Web (WCAG) 2.0, para que una página web sea conforme con las WCAG 2.0, deben satisfacerse todos los requisitos de conformidad siguientes:

1. ***Nivel de conformidad***. Se recoge un conjunto de criterios de conformidad, redactados en forma de enunciados verificables sobre el contenido web, y que son utilizados para verificar la adecuación a la accesibilidad de un sitio web. Se distinguen tres niveles de conformidad que definen el grado de accesibilidad, uno de ellos se debe satisfacer por completo.
   - Nivel A: para lograr conformidad con el nivel A (el mínimo), la página web satisface todos los criterios de conformidad del nivel A, o proporciona una versión alternativa conforme. Se deben satisfacer 25 criterios.:
   - Nivel AA: para lograr conformidad con el nivel AA, la página web satisface todos los criterios de conformidad de los niveles A y AA, o se proporciona una versión alternativa conforme al nivel AA. Se deben satisfacer 13 criterios.
   - Nivel AAA: para lograr conformidad con el nivel AAA, la página web satisface todos los criterios de conformidad de los niveles A, AA y AAA, o proporciona una versión alternativa conforme al nivel AAA. Se deben satisfacer 23 criterios.

2. **Páginas completas**. La conformidad se aplica a páginas web completas, y no se puede alcanzar si se excluye una parte de la página.

3. **Procesos completos**. Cuando una página web es parte de una serie de páginas web que presentan un proceso, todas las páginas en ese proceso deben ser conformes con el nivel especificado o uno superior.

4. ***Uso de tecnologías exclusivamente según métodos que sean compatibles con la accesibilidad***. Para satisfacer los criterios de conformidad solo se depende de aquellos usos de las tecnologías que sean compatibles con la accesibilidad.

5. ***Sin interferencia***. Si las tecnologías se usan de una forma que no es compatible con la accesibilidad, o está usada de una forma que no cumple los requisitos de conformidad, no debe impedir a los usuarios acceder al contenido del resto de la página.

Principios de la accesibilidad

Principio 1. Perceptible. La información y los componentes de la interfaz de usuario deben ser mostrados a los usuarios de tal manera que pueda ser entendida. Para ello se establecen cuatro directrices:

- Directriz 1.1. Texto alternativo. Proporcionar texto alternativo para el contenido que no sea textual (caracteres grandes, lenguaje Braille, lenguaje oral, símbolos o lenguaje más simple).

- Directriz 1.2. Contenido multimedia dependiente del tiempo. Proporcionar acceso a los elementos multimedia (audio y/o video) a través de subtítulos y otras alternativas.

- Directriz 1.3. Adaptable. Crear contenido que pueda ser presentado de diferentes formas sin perder información o estructura.

- Directriz 1.4. Distinguible. Facilitar a los usuarios ver y escuchar el contenido, incluyendo la distinción entre lo más y lo menos importante.

Principio 2. Operable. Los componentes de la interfaz de usuario y la navegación deben ser manejables.

- Directriz 2.1. Teclado accesible. Poder controlar todas las funciones desde el teclado.

- Directriz 2.2. Tiempo suficiente. Proporcionar tiempo suficiente a los usuarios para leer y utilizar el contenido.

- Directriz 2.3. Ataques epilépticos. No diseñar contenido que pueda causar ataques epilépticos.

- Directriz 2.4. Navegación. Aportar formas para ayudar a los usuarios a navegar, a buscar contenido y a determinar dónde están estos.

Principio 3. Comprensible. La información y las operaciones de usuarios deben ser comprensibles.

- Directriz 3.1. Legible. Hacer contenido de texto legible y comprensible.

- Directriz 3.2. Previsible. Hacer la apariencia y la forma de utilizar las páginas web previsibles.

- Directriz 3.3. Asistencia a la entrada de datos. Los usuarios de ayuda evitarán y corregirán errores.

Principio 4. Robustez. El contenido ha de ser robusto para que pueda ser interpretado por una gran variedad de agentes de usuario, incluyendo tecnologías de asistencia.

- Directriz 4.1. Compatible. Maximizar la compatibilidad con los agentes de usuario actuales y futuros, incluyendo tecnologías de asistencia.

7.3.2. Tipos de discapacidad

Como hemos visto en el apartado anterior, existen diversas casuísticas de usuarios en función de sus capacidades cognitivas, tecnológicas o culturales. Otro de los puntos que deben tener en cuenta son las discapacidades: en este punto veremos algunas de las más importantes, relativas a lo que el diseño de interfaces se refiere, objeto principal de este libro. Hablemos de discapacidades visuales, motrices y auditivas. En el siguiente apartado se verán diferentes tipos de tecnologías aplicadas a la accesibilidad.

A) Discapacidad visual

Se define como discapacidad visual la dificultad que presentan algunas personas para participar en actividades propias de la vida cotidiana, que surge como consecuencia de la interacción entre una dificultad específica relacionada con una disminución o pérdida de las funciones visuales y las barreras presentes en el contexto en que se desenvuelve la persona. Se distingue entre ceguera o deficiencia visual: en el primer caso se produce una limitación total de la función visual, mientras que en el segundo no llega a ser total, por lo tanto, las medidas que se deben tomar en cada caso son diferentes. Por ejemplo, para personas sin una ceguera total, una posible medida sería el diseño de una versión de la interfaz con los tamaños más grandes que en la versión original.

A la hora de diseñar el sitio web, es deseable que el diseño de la aplicación o aplicación web accesible se centre en evitar las siguientes barreras de acceso para las personas con discapacidad visual:

1. ***Imposibilidad de acceder al contenido, o de operar con la aplicación, desde el teclado del ordenador.***
   Para garantizar la accesibilidad de una aplicación es necesario contemplar la funcionalidad y operatividad total a través del teclado del ordenador. Uno de los casos más comunes es el envío de un formulario a través de un botón que solo puede ser activado mediante el ratón o un menú que solo se despliega al pasar el puntero del ratón sobre él, para estos casos debe existir la posibilidad de asignar teclas de acceso a cualquier enlace o control de formulario que proporcione un atajo mediante el teclado. Ahora bien, es importante tener en cuenta que la creación de estos atajos no choque con los ya existentes en cualquier sistema; será difícil encontrar un amplio abanico de atajos de teclas disponibles, por lo que se recomienda utilizar teclas de acceso rápido a elementos de la página web muy frecuentes o de difícil localización.

2. ***Ausencia de textos alternativos para los elementos no textuales.*** Se deben proporcionar accesos complementarios a los elementos multimedia, puesto que su ausencia puede provocar que los usuarios no puedan acceder al contenido. En el caso de los usuarios con discapacidad visual, al utilizar vídeos será conveniente que se describa lo que se muestra, habitualmente a través de un texto que es reproducido por audio, consiguiendo así que estos usuarios tengan una experiencia satisfactoria al utilizar la aplicación.

3. ***Formularios y tablas de datos complejos y difíciles de interpretar correctamente***. Cuando se crean formularios es especialmente necesario cuidar ciertos aspectos de accesibilidad para poder garantizar una correcta interpretación de los contenidos a los usuarios de lectores de pantalla.

   Uno de los elementos utilizados por los usuarios con discapacidad visual son los lectores de pantalla, que consisten en un software que "lee y explica" lo que hay en la pantalla. En el caso de los formularios, el software debe ser capaz de identificar claramente de qué tipo de control se trata, de indicar su estado y de anunciar la etiqueta correspondiente a dicho control. Algunos de los más conocidos para el uso de contenido web son:

   - **BrowseAloud**. Lector de pantalla destinado específicamente a leer el contenido de las páginas web. Está disponible para Windows y para Mac.

   - **WebAnyware**. Lector de pantalla en la web.

   - **vozMe**. Explica cómo incorporar el componente en Wordpress, Blogger o cualquier otra página web. Permite elegir entre una voz masculina o femenina y también permite descargar un fichero MP3 con el audio.

4. ***Utilización inadecuada de elementos estructurales en las páginas o falta de estructuración en sus contenidos***: ausencia de encabezados de sección, definiciones de listas, agrupaciones de controles.

   La manera en que se estructura el contenido de una página web es fundamental en lo que se refiere a su accesibilidad. Es muy importante identificar correctamente los elementos de estructura básicos como encabezados, listas, párrafos, tablas de datos, etc.

   - ***Sitios con pobre contraste de color o con información basada en el color***.

   - ***Presencia de captchas*** en los que no se aporta solución accesible. En el caso de las imágenes de texto visualmente distorsionadas, que se usan como mecanismos de control destinados a distinguir a un humano de una máquina o programa de ordenador, se deben proporcionar distintos métodos alternativos para acceder a la información, adaptados a diferentes capacidades sensoriales.

La evolución tecnológica ha creado y adaptado diversos dispositivos para contribuir a que cualquier usuario con una discapacidad visual sea capaz de utilizar un sitio web de igual forma que lo haría si su capacidad visual fuera la misma que la de un usuario sin esta dificultad. En este caso, se centra en el desarrollo de dispositivos de salida, como sucede con las pantallas o las impresoras.

A lo largo de este punto hemos hablado de los lectores de pantallas, indicados para las personas con ceguera total. Para aquellas que tienen una dificultad visual parcial existen los ampliadore de pantallas, esto son programas que permiten ampliar el texto y las imágenes, incluso existen navegadores especiales que leen las páginas web y reconocen la voz del usuario para navegar por ellas.

B) Discapacidad auditiva

La discapacidad auditiva se define como el déficit total o parcial en la percepción del sonido, que se evalúa por el grado de pérdida de la audición en cada oído. Se pueden distinguir entre las personas hipoacúsicas, que son las que presentan una deficiencia parcial, es decir, cuentan un resto auditivo que pueden mejorar a través de audífonos.Y, por otro lado, las personas sordas, las cuales presentan una deficiencia total.

La principal barrera que se destaca en este caso es el incipiente uso elementos audiovisuales en la web (vídeos, animaciones, sonidos, etc.). Para que estos puedan ser accesibles para personas con discapacidad auditiva deberán presentar una alternativa de texto.

C) Discapacidad motora

Una persona con una discapacidad motora es aquella que sufre de una manera duradera y frecuentemente crónica una afección más o menos grave del aparato locomotor que supone una limitación de sus actividades en relación con el promedio de la población. Se debe conocer también que esta cubre todos los trastornos que pueden causar deterioro parcial o total de las habilidades motoras, incluyendo la parte superior y/o inferior del cuerpo.

Como consecuencia de esto, en términos relativos al diseño podemos encontrar personas con dificultades para llevar a cabo algunas tareas informáticas, como utilizar un ratón, mover un puntero, utilizar una pantalla táctil, pulsar dos teclas al mismo tiempo o mantener pulsada una tecla, incluso pueden ser incapaces de utilizar un teclado y no poder introducir datos.

Las principales barreras de accesibilidad que se pueden producir en el diseño de una interfaz son:

1. ***Imposibilidad de interaccionar adecuadamente con la página desde el teclado u otros dispositivos de entrada***. Para facilitar la interacción se recomienda usar dispositivos de

entrada diseñados especialmente para ser utilizados con el ordenador en general y la navegación en particular. Por ejemplo, teclados con teclas grandes, teclado convencional con colcha, ratón de bola grande o trackball, ratón de botón, entre otros.

2. ***Enlaces gráficos y otros elementos accionables que no están etiquetados correctamente*** y no son accesibles a los reconocedores de voz. De nuevo, al igual que ocurre con las discapacidades visuales en el caso de los lectores, en este serán de relevante importancia los reconocedores de voz, a través de los cuales los usuarios serán capaces de realizar las operaciones deseadas, ahora bien, para ello el diseño de los elementos deben ser fácilmente accionables a través de este tipo de dispositivos.

Al igual que en el primer caso, el desarrollo de dispositivos de salida ha mejorado la accesibilidad, en este caso, será el software o dispositivos de entrada los que recojan nuestra atención. Por ejemplo:

- ***Software de reconocimiento de voz***. Software indicado para aquellos casos en los que no se puede hacer uso del teclado o del ratón.

- ***Trackball***. Consiste en un dispositivo similar a un ratón, pero que no requiere del desplazamiento de este para funcionar, sino que el desplazamiento se lleva a cabo utilizando una bola que al girar permite el desplazamiento del ratón por la pantalla.

- ***Cámaras web***. A través del reconocimiento facial, permiten traducir el movimiento de la cara o los ojos en movimiento del ratón por la pantalla.

7.4. Análisis y verificación de la usabilidad

Es fundamental tener la capacidad de analizar y verificar qué grado de usabilidad tiene una determinada aplicación y su interfaz en base a los objetivos y necesidades de los usuarios. Esta información permitirá realizar posibles mejoras o modificaciones en el producto, en aras de aumentar la usabilidad del mismo.

Este proceso de análisis y verificación de la usabilidad se suele llevar a cabo en la fase de evaluación de un proyecto.

Existen diversos métodos que se utilizan comúnmente para este tipo de tareas. Á continuación, se definen algunos de los métodos más ampliamente utilizados en el desarrollo de aplicaciones web.

7.4.1. Método por inspección. Evaluación heurística

Este tipo de método se lleva a cabo por profesionales expertos en usabilidad, que se dedican a analizar de manera completa, identificando posibles problemas que es necesario corregir. La base de este método es tanto la propia experiencia de los expertos que evalúan el sitio como los códigos de buenas prácticas o guías existentes para detectar ciertos principios relacionados con la usabilidad.

Algunos de estos principios pueden ser los siguientes:

- Cumplimiento de directrices de accesibilidad.

- Utilización del mismo lenguaje entre aplicación y usuario.

- Información aportada al usuario por parte del sistema sobre el proceso que está llevando a cabo, es decir, sobre lo que está sucediendo. Un ejemplo típico de este hecho es cuando un determinado usuario quiere acceder a un contenido audiovisual que ofrece el portal, y el portal le informa que está procediendo a la carga en búfer del contenido.

- Fomentar el control de la interfaz de la aplicación por parte del usuario. Por ejemplo, es importante que el usuario tenga la capacidad de poder dar marcha atrás en alguna acción que haya llevado a cabo, ya que puede tratarse de un error. Así mismo, es importante facilitar y orientar al usuario para poder resolver un error determinado, por ejemplo, a la hora de completar un campo de un formulario.

- Incluir documentación de ayuda que pueda ser consultada en un momento por el usuario.

- Estructura adecuada de la información mostrada en la aplicación.

- Inclusión de elementos multimedia y adecuación de los mismos a la temática del sitio.

- Calidad adecuada del contenido en cuanto a lenguaje y redacción se refiere.

La gran ventaja de este método respecto a otros es que se puede desarrollar de una manera sencilla, rápida y eficaz en un plazo breve de tiempo, aunque su coste puede ser mayor.

7.4.2. Método de test con usuarios

Al contrario que en el método por inspección, este método se basa en el análisis de una aplicación a través de un grupo de usuarios reales, de manera que dichos usuarios puedan detectar problemas de utilización o bien plantear opciones de mejora.

Es importante tener en cuenta que este método puede implementarse una vez que la aplicación ya se encuentra en la fase de producción para que los usuarios tengan una visión completa de la funcionalidad de la aplicación, pero también puede llevarse a cabo durante las fases de diseño e implementación de la misma. Este hecho es muy importante, ya que es menos costoso corregir problemas durante estas fases del diseño que hacerlo en la fase de producción.

Este no tiene por qué considerarse como un método contrapuesto al método de inspección, sino que ambos pueden complementarse en el análisis de la usabilidad.

Así mismo, la gran ventaja de este tipo de métodos es que los resultados son más fiables y, además, se profundiza en el descubrimiento de errores de diseño relacionados con la usabilidad.
Sin embargo, hay que tener en cuenta el elevado coste que puede suponer, ya que para llevarlo a cabo será necesario disponer de varios usuarios, así como ubicarlos en una zona adecuada para realizar su tarea, prestándoles las herramientas necesarias para ello.

7.5. Análisis y verificación del proceso de desarrollo de interfaces

A la hora de implementar un determinado proyecto, se deben establecer unas fases o procesos para facilitar la consecución del objetivo. Es decir, es necesario aplicar una serie de procedimientos, técnicas y métodos específicos para el objeto que se persigue. De esta forma, para la realización de una interfaz y su aplicación se establecen las siguientes fases:

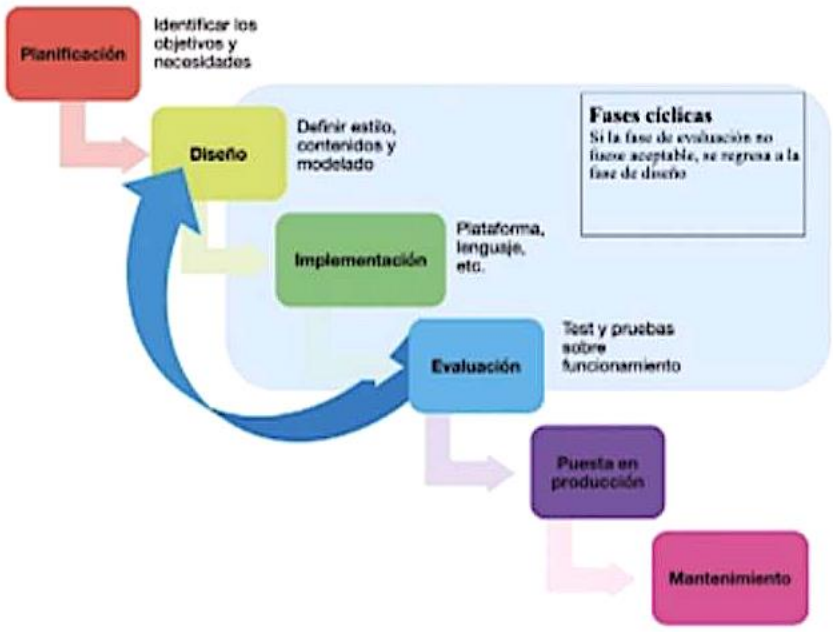


Figura 5.8 – Fases de análisis y planificación de un sitio web

7.5.1. Fase de planificación

Todo proyecto que se precie, sea del ámbito que sea, debe planificarse con anterioridad a su inicio, independientemente del objetivo que persiga. En esta fase es necesario fijar principalmente los objetivos que se pretenden alcanzar, además de los medios a través de los cuales se llegará a los mismos.

De manera más concreta, en el caso de una aplicación se deberán tener en cuenta detalles como los recursos profesionales y técnicos necesarios, tipo de almacenamiento, costes. Además, será necesario sentar las bases de los contenidos que tendrá el sitio y de cómo se dispondrán los mismos para proporcionar una experiencia grata al usuario.

Por otro lado, en esta fase es especialmente crítico que se obtenga la mayor cantidad posible de información acerca del usuario, ya sea a través de encuestas, entrevistas, reuniones, etc., para identificar necesidades, objetivos, comportamientos.

7.5.2. Fase de diseño

Durante esta fase se implementará el aspecto y funcionalidades que tendrá la aplicación, teniendo siempre en cuenta la importancia de la usabilidad, así como toda la información obtenida durante la fase de planificación.

Es fundamental que durante esta fase el diseñador valore los tipos de usuario que puede tener la aplicación para considerar sus necesidades, limitaciones y habilidades, y adaptar el desarrollo de la aplicación a las mismas.

En el diseño de aplicaciones con varias pantallas y elementos de navegación es muy aconsejable realizar de manera previa un esquema de contenidos y organización del sitio, identificando enlaces entre menús o ventanas. También será aconsejable documentar de manera adecuada todas las acciones que se realicen para facilitar su comprensión a diseñadores externos que puedan requerir esa información llegado el caso.

7.5.3. Fase de implementación

Durante esta fase, se realizará el diseño planteado en la fase anterior, obteniéndose de esta forma las primeras versiones operativas de la aplicación. Es fundamental en este caso el haber considerado adecuadamente los lenguajes de programación que se utilizarán, así como las plataformas de desarrollo.

Es muy aconsejable tener la opción de realizar versiones previas o prototipos de la aplicación para poder analizar sobre las mismas la experiencia del usuario en cuanto a usabilidad y accesibilidad se refiere.

7.5.4. Fase de evaluación

Se trata sin duda de una de las etapas más críticas del proceso, dado que sobre la misma se tomará la decisión de pasar la aplicación a producción o, por el contrario, qué aspectos deben modificarse y cómo se llevará a cabo. Por ello, se trata de una fase a partir de la cual se puede volver a la fase de diseño o bien avanzar a la fase de puesta en producción.

Nota:
En este tipo de diseños, el usuario debe ser el centro de todo el proceso. Todas las fases deben estar enfocadas a atender los objetivos del mismo, por lo que es necesario involucrar a los usuarios desde la fase de planificación para poder garantizar su satisfacción y atender adecuadamente sus necesidades y requerimientos. También será de suma importancia el involucrar a los usuarios durante la fase de evaluación para así comprobar sus experiencias de uso e implementar las posibles mejoras que pudieran ser necesarias.

7.5.5. Fase de puesta en producción

Como su propio nombre indica, durante esta fase se lleva a cabo la puesta a disposición de los usuarios de la aplicación, una vez ha sido comprobado y analizado su funcionamiento en las etapas anteriores del proceso.

7.5.6. Fase de mantenimiento y seguimiento

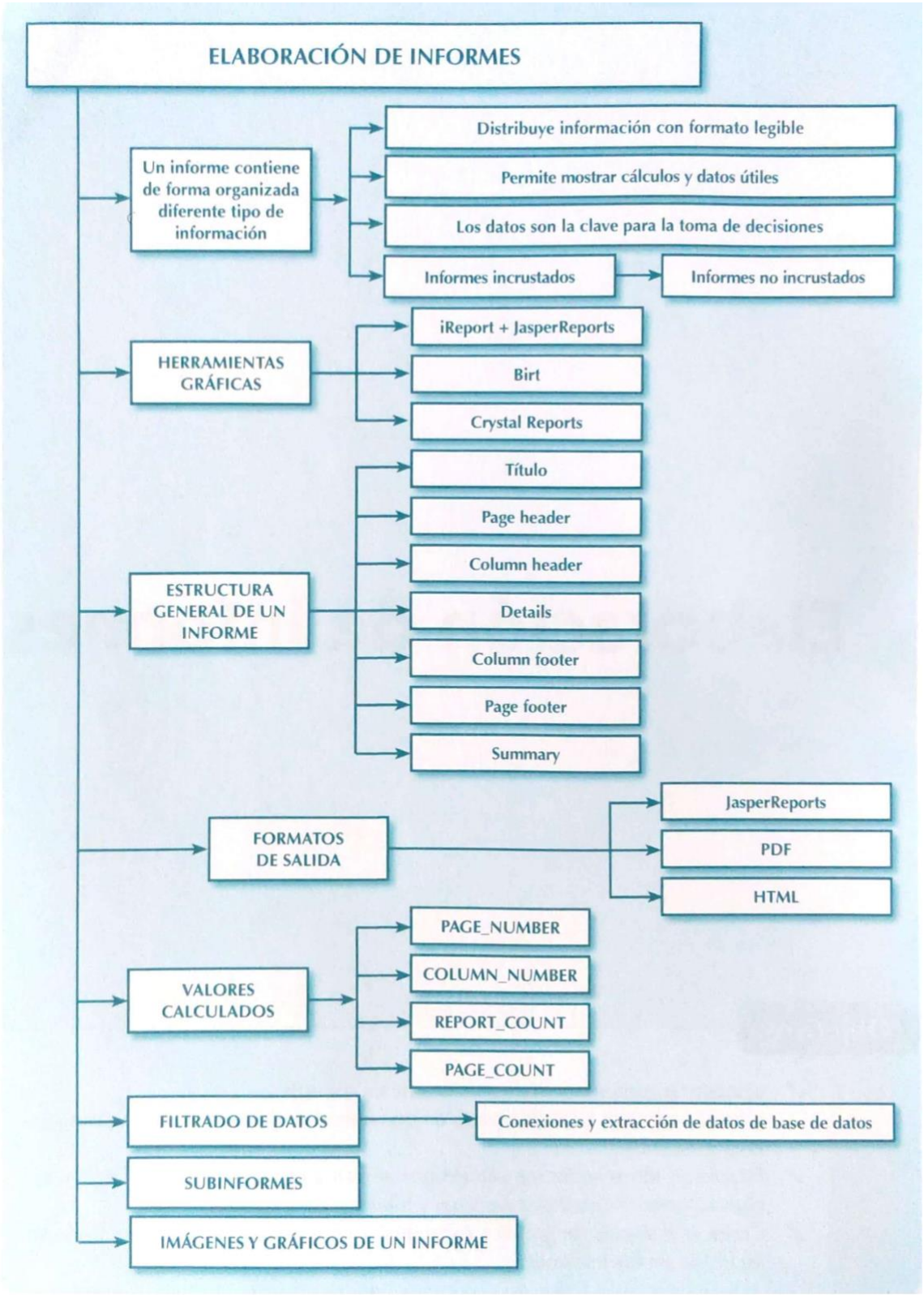
Se trata de la última fase del proceso, aunque debe tratarse como una fase crítica. Se trata de una fase fundamental, debido a que la tecnología evoluciona continuamente, y con ella las necesidades y posibilidades de los usuarios, por lo que es muy importante adaptar y mejorar el sitio a partir de esto.

UD5. Creación de informes mediante uso y evaluación de herramientas gráficas

Objetivos

- Conocer y comprender la estructura de un informe.
- Generar informes básicos a partir de una fuente de datos mediante herramientas y asistentes.
- Establecer filtros sobre los valores que se van a presentar en los informes. Incluir valores calculados, recuentos y totales
- Conocer los tipos de gráficos existente y comprender la importancia de su inclusión en los informes.

Mapa conceptual



Glosario

Filtrado de datos. Proceso por el cual se obtienen los datos que resultan de interés para una determinada aplicación, a partir de un conjunto de datos global. Se suelen utilizar consultas tipo SELECT para obtener los mismos a partir de grandes volúmenes de datos.

Herramienta gráfica. En el entorno del desarrollo de aplicaciones, una herramienta gráfica es una aplicación que se utiliza con un fin determinado mediante una interfaz.

Informe. Tipo de documento que permite ser elaborado en diferentes formatos, que contiene de manera organizada información acerca de un campo concreto, como puede ser el de una aplicación informática.

Subinforme. Son informes incluidos dentro de otros informes, de manera jerarquizada. Se suelen utilizar con el fin de mejorar la comprensión y legibilidad de la información, así como para optimizar la estructura del informe en su conjunto.

Valor calculado. Se trata de valores utilizados habitualmente en la confección de informes y cuyo resultado se basa en la utilización de variables, que permiten realizar su cálculo. Su valor puede ir modificándose conforme se avanza en la elaboración del informe.

Variables de usuario. Como su propio nombre indica, son variables que pueden ser creadas por el usuario. Habitualmente se trata de los campos que son origen de datos vinculados al informe.

Variables predefinidas. Son variables creadas por defecto en la herramienta de generación de informes, por lo que siempre van a estar disponibles.

5.1. Informes en la aplicación

Un informe es un tipo de documento que contiene de manera organizada información acerca de un campo concreto, como puede ser el de una aplicación informática. Un informe necesita de un origen del que extraer información necesaria para su confección.

Los informes que se obtienen de una aplicación posibilitan la extracción y el análisis de muy distintos tipos de información, como pueden ser los datos relacionados con el uso que se le está dando a la aplicación, datos acerca de las conexiones que tiene la aplicación, sobre la utilización de las mismas, etc. Como es fácil deducir, el uso que se puede hacer de estos informes es muy variado, y depende en gran medida de la aplicación y del objetivo que se desee satisfacer.

Mediante la creación de informes es posible presentar los datos que se obtienen de múltiples maneras, ya sea utilizando plantillas ya existentes o bien a través de otras herramientas que permiten generar informes de una manera personalizada.

El proceso de creación de informes tiene una gran importancia en muchos campos, por lo que no es un proceso trivial. Durante el mismo existe una serie de tareas o recomendaciones que conviene tener en cuenta, como las que se muestran en la siguiente imagen:

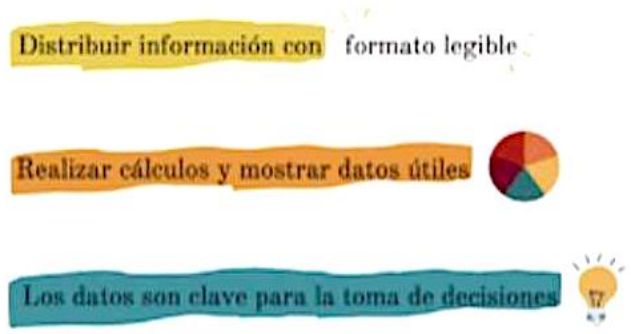


Figura 9.1 Recomendaciones para la creación de informes.

La elaboración de un informe depende en gran medida del destinatario final, así como de los objetivos que se deseen cumplir. De esta manera, el enfoque y contenido del mismo puede variarse en función de estas casuísticas.

5.1.1. Informes incrustados

Los llamados informes incrustados se distinguen porque son generados de manera directa desde la aplicación en concreto. Un ejemplo de ellos podrían ser los informes que se generan desde herramientas integradas en un IDE concreto.

El programa o aplicación en concreto del que se desean obtener contiene típicamente un módulo en concreto para facilitar estos informes, que pueden estar relacionados con la utilización de la aplicación, los tiempos de respuesta, las alarmas generadas, etc., en función de cada aplicación concreta.

5.1.2. Informes no incrustados

Al contrario que en el caso anterior, los informes no incrustados se elaboran de manera externa a la aplicación en concreto, es decir, se crear a partir de aplicaciones específicas para ello, que no están integradas con la aplicación concreta ni con su entorno de desarrollo.

5.2. Herramientas gráficas

Existen diversas herramientas gráficas que facilitan la creación de informes, ya que permiten al usuario diseñar y generar los mismos de una manera más intuitiva y sencilla. Se trata de herramientas muy potentes que facilitan en gran medida la elaboración de los informes. En este capítulo analizaremos algunas de las más importantes de las que están relacionadas con los entornos de desarrollo de aplicaciones y que incluso pueden ser integradas en entornos de desarrollo como Eclipse.

- **iReport + JasperReports:** se trata de dos herramientas que actúan de manera conjunta. En el caso de JasperReports, se utiliza para encapsular los informes en los entornos de desarrollo, mientras que iReport construye la interfaz gráfica, que posibilita una construcción más intuitiva y sencilla del informe. Son dos de las herramientas más conocidas y utilizadas, especialmente en entornos de desarrollo Java, integrándose con otras herramientas como Eclipse o NetBeans.

- **Birt:** se trata de una herramienta open source (es decir, de código abierto), que posibilita crear informes de una manera dinámica y ágil, en entornos empresariales. Se suele integrar con entornos de desarrollo como Eclipse, por lo que es muy utilizada en el desarrollo de aplicaciones informáticas en Java.

- **Crystal Reports:** esta herramienta de inteligencia empresarial se utiliza para crear informes de forma dinámica. Se utiliza sobre todo en el entorno de desarrollo Microsoft Visual Studio.

Nota:
Las herramientas gráficas facilitan la creación de informes de una manera sustancial, optimizando los resultados y reduciendo las horas de trabajo necesarias para su elaboración. En función del tipo de informe y de aplicación, se seleccionará una u otra, según las características y preferencias del usuario.

5.3. Estructura general de un informe. Secciones, encabezados y pies

Los informes suelen tener una estructura general sobre la que se aplica el formato del documento y se integran los datos. Esta estructura está construida mediante diferentes secciones, que pueden utilizarse de manera genérica o bien modificarse en función del objetivo y del tipo de informe.

En la siguiente imagen se muestra una posible plantilla de informe, sobre la cual se deben incluir los diferentes datos en función del caso concreto.

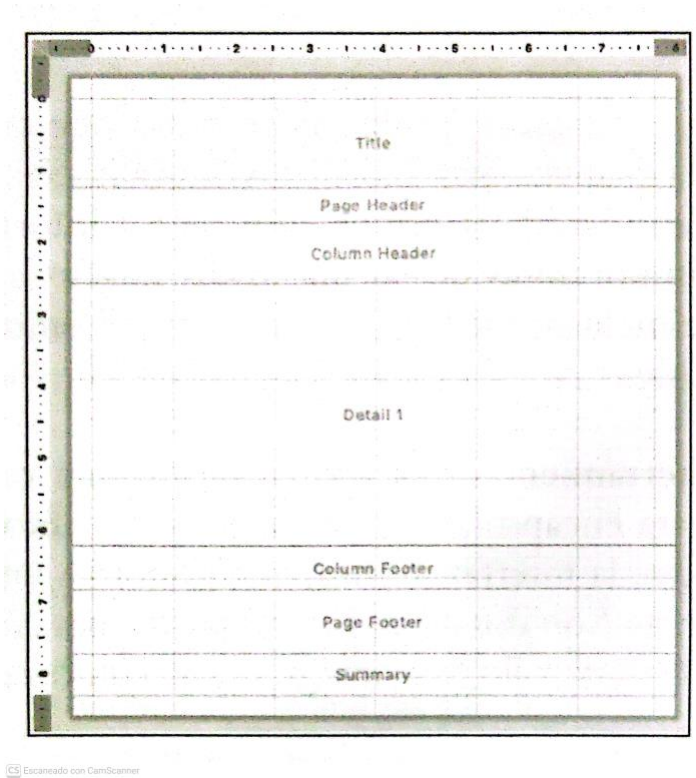


Figura 5.2 - Plantilla de informe con las diferentes

Nota:
En el momento de realizar un informe es muy importante pensar a qué tipo de usuario va dirigido y qué información es la que puede resultarle más relevante, para poder el foco en la misma.
En todo caso, se debe evitar la información redundante o aquella que pueda resultar confusa.

De las diferentes secciones que se pueden apreciar en la imagen, podemos distinguir las secciones de encabezado (Título, Page Header, Column Header) y de pies (Column Footer, Page Footer y Summary).

Las características de los diferentes elementos se muestran en el siguiente cuadro.

Cuabro 9.1 - Secciones habituales en un informe

Título	Como su nombre indica, en esta sección se indica el nombre del informe, que debe ser claro y conciso para dar una idea general de su contenido.
Page Header	Contiene datos generales sobre el informe, como el autor, la fecha de generación.
Column Header	Suele ser habitual la inclusión de datos en forma de columnas en los informes, bajo las que se organiza la información que se va a mostrar. Las etiquetas o títulos de dichas columnas se incluyen en esta sección.
Details	Esta es la sección del informe en la que se incluyen todos los detalles relacionados con los parámetros y datos que constituyen el cuerpo del mismo. Es decir, es la sección que incluye los datos completos del informe.
Column footer	Se utilizan habitualmente para incluir datos relacionados con el contenido de cada columna del informe.
Page footer	De manera similar al encabezado, contiene datos generales sobre el informe, como pueden ser los números de página, por ejemplo.
Summary	En esta sección se incluye un resumen de los datos presentados en el informe, ya sea en forma de conclusiones o en forma de valores calculados, recuentos u otras operaciones.

5.4. Formatos de salida

Los formatos en los que se pueden construir los informes son muy variados, y tienen una dependencia directa de la herramienta que se utilice para ello. Por ejemplo, si se utiliza una hoja de cálculo, lo más probable es que el formato del mismo sea del tipo .xlsx o .csv. Es muy importante tener en cuenta que es tan importante la información que se muestra en el informe como la forma en la que se hace.

La gran cantidad de formatos posibles que soportan las herramientas para la creación de informes es una de sus principales ventajas, ya que aporta un gran abanico de opciones que se adaptan a prácticamente cualquier necesidad, posibilitando una adaptación del formato de salida al objetivo que se desea cubrir mediante la elaboración del informe.

Figura 5.3
Formatos de salida más utilizados

Por ejemplo, la herramienta **iReport** permite seleccionar entre múltiples opciones de formatos de salida para mostrar y generar el informe diseñado con los datos y los cálculos escogidos.

Otro ejemplo, si se utiliza la aplicación **JasperReports** para la realización del informe, se cuenta con una gran cantidad de formatos de salida disponibles, tal y como se puede observar en la figura 5.4:

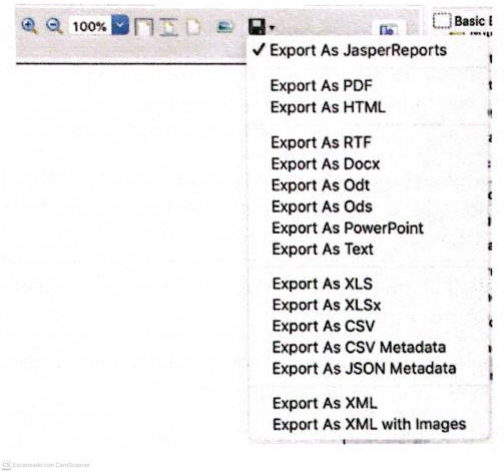


Figura 5.4 - Formatos de salida disponibles en la herramienta JasperReports.

El hecho de construir un informe en un formato determinado posibilita su integración con otros usos. Por ejemplo, es posible generar un informe como PowerPoint para poderlo integrar en una presentación ejecutiva sobre la aplicación en concreto. Otro ejemplo posible sería extraer el informe como PDF para facilitar su envío a través del correo electrónico.

Es también muy habitual utilizar como salida el formato CSV, ya que permite incluir una gran cantidad de datos de una manera muy eficiente, permitiendo a su vez ser procesado por algunas de las principales aplicaciones de hojas de cálculo, como pueden ser Numbers o Excel.

Nota:
Es muy importante el formato que se seleccione para la elaboración del informe. Para ello, es preciso tener en cuenta la herramienta que se va a utilizar para ello, así como la manera en que será visualizado el documento. Es usual que una misma herramienta pueda generar (o interpretar) archivos de diferente formato.

5.5. Valores calculados

Es muy habitual que en muchas ocasiones se necesite trabajar con algún valor concreto, que sea resultado de algún cálculo sobre otros campos o datos. A esto se lo conoce como valores calculados, y su uso se basa en la utilización de variables. Son parámetros de uso recurrente en la confección de informes.

La utilización de variables en la elaboración de informes es muy útil, dado que el valor de las mismas puede ir cambiando conforme se va creando el informe, y, además, pueden evaluarse en distintos momentos o ubicaciones del mismo. De esta manera, el uso de variables es fundamental para poder calcular determinados valores, y poderlos incluir a continuación en el informe.

Existen dos tipos de variables:

a) **Variables de usuario**. Como su propio nombre indica, son variables que pueden ser creadas por el usuario. Habitualmente se trata de los campos que son origen de datos vinculados al informe.

b) **Variables predefinidas**. Son variables creadas por defecto en la herramienta de generación de informes, por lo que siempre van a estar disponibles. Algunas de ellas se muestran en el siguiente cuadro.

Cuadro 5.2 - Variables predefinidas

Nombre	Descripción
PAGE_NUMBER	Almacena el número de páginas del informe (tipo Integer).
COLUMN_NUMBER	Indica el número de columnas del informe (tipo Integer).
REPORT_COUNT	Indica el número de registros para una consulta (Integer).
PAGE_COUNT	Proporciona el número de registros mostradas en cada página (Integer).

5.5.1. Numeración de líneas. Recuentos y totales

Los valores calculados se suelen utilizar a modo de resumen de los datos que contiene un informe. Por lo tanto, hay algunas operaciones comunes que se realizan con los mismos, como las siguientes:

- Numeración de líneas. Suele ubicarse en la zona de “Details” del informe y, como su nombre indica, se utiliza para obtener el número de líneas que tiene el informe. El valor de esta variable se irá incrementando conforme se añada contenido al mismo.

- Recuentos y totales. Se trata de operaciones que se suelen utilizar sobre los valores de una columna. Por ejemplo, operación suma aplicada sobre todos los valores de una determinada columna para obtener el total, u operación recuento sobre los valores de una columna para obtener el número de registros totales.

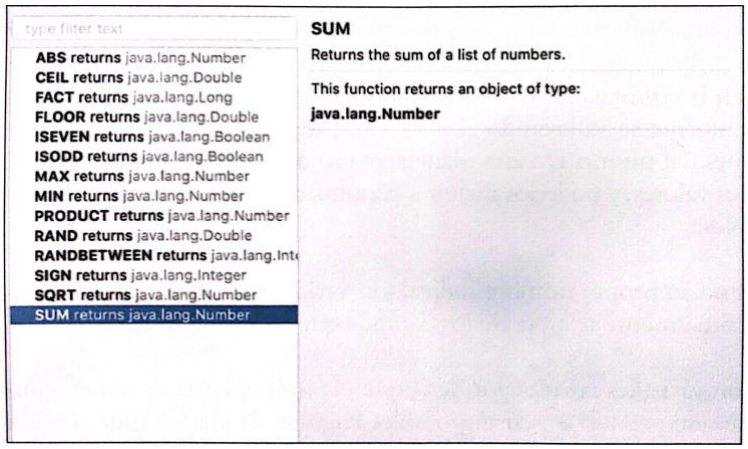


Figura 5.5 - Operaciones más utilizadas sobre columnas

5.6. Filtrado de datos. Conexión a bases de datos y diseño de consultas

Para poder crear informes robustos y completos es fundamental considerar el origen de los datos, es decir, la base de datos que contiene la información con la que vamos a trabajar. En función del tipo de base de datos que vayamos a utilizar, puede ser necesario realizar algún tipo de configuración previa para poder obtener datos de esta.

En el caso de utilizar bases de datos MySQL u Oracle, es muy posible que así sea. Las herramientas como Eclipse o Netbeans contienen programas y librerías que se utilizan como conectores, y que permiten trabajar con bases de datos de este tipo.

5.6.1. Diseño de consultas

Las consultas sobre lenguaje SQL (el utilizado de manera habitual para el acceso a información de bases de datos) se realizan mediante el uso de la sentencia SELECT. Esta sentencia puede construirse en base a diferentes opciones y parámetros, que variarán según la información que se desee obtener, y que puede estar ubicada en una o varias tablas de la base de datos.

La sintaxis de la misma es la siguiente:

```
SELECT [* | DISTINCT] <campos>
FROM <tablas>
[WHERE <condicion> [AND | OR <condición>]]
```

```
[GROUP BY <nombre_campo>]
[HAVING <criterios de agrupación>]
[ORDER BY <nombre_campo>|<índice_campo> [ASC|DESC]]];
```

Figura 5.6 - Estructura de la sentencia SELECT en SQL.

En primer lugar, se pueden indicar las columnas de la tabla que se desea obtener, o el símbolo * si se desean obtener todas ellas. La sentencia FROM ubicada a continuación permite indicar el nombre de la tabla o tablas a partir de las cuáles deseamos obtener la información.
Podemos decir que estas son las cláusulas básicas, que se utilizan en prácticamente cualquier consulta de tipo SELECT.

A continuación de estas, existe un grupo de cláusulas adicionales, que se utilizan de manera totalmente opcional, y que se muestran entre corchetes en la imagen anterior. A partir de estas cláusulas es sencillo realizar operaciones de filtrado de datos, es decir, podemos utilizar estas cláusulas para discriminar las características de los datos que deseamos obtener mediante la consulta. Se basan en el uso de sentencias condicionales, que son de aplicación sobre los datos que necesitamos obtener.

De esta manera, es fácil deducir que estas cláusulas pueden facilitarnos la obtención de los datos que necesitamos incluir en nuestro informe, siendo un procedimiento muy habitual en la extracción de información de bases de datos mediante SQL.

La cláusula WHERE, por ejemplo, se utiliza para indicar una o varias condiciones que deben cumplir los datos que se desean obtener, utilizándose por tanto para filtrar los datos que nos interesan del conjunto total de datos existentes. La sintaxis que seguiría una expresión de tipo SELECT que utilice cláusulas FROM y WHERE se muestra en la siguiente imagen:

```
SELECT [* | DISTINCT] <campos>
FROM <tablas>
[WHERE <condición> [AND | OR <condición>]]
```

Figura 5.7 - Estructura de sentencia SELECT con cláusulas FROM y WHERE.

Este es el tipo de consulta más habitual sobre una base de datos de tipo SQL. A continuación, se muestra un ejemplo práctico de una sentencia de este tipo, a través del cual se mostrarían todas las filas de la tabla TablaDatos que cumplan la condición indicada en la cláusula WHERE:

```
SELECT * FROM TablaDatos
WHERE ID = $P[valor_filtrado]
```

Figura 5.8 - Ejemplo de consulta aplicando filtrado de datos.

5.7. Subinformes

Los subinformes se pueden definir como informes incluidos dentro de otros informes, de manera jerarquizada. Se suelen utilizar con el fin de mejorar la comprensión y legibilidad de la información, así como para optimizar la estructura del informe en su conjunto.

Al mismo tiempo, los subinformes proporcionan un método para trabajar con datos relacionales, ya que posibilitan la inclusión de datos en un mismo informe que han sido obtenidos a través de varias consultas. Todo ello de una manera más clara y organizada, facilitando así la comprensión de la información por parte del usuario.

En la siguiente imagen se muestra un ejemplo de subinforme, cuyos datos son producto de diferentes consultas sobre el origen de los datos

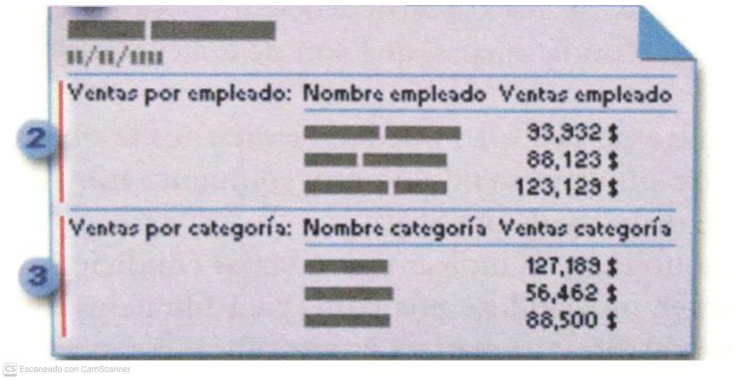


Figura 5.9 - Ejemplo de subinforme en un informe principal.

Basándonos en el ejemplo anterior, el informe principal contendrá dos subinformes, el primero con la información relativa a las ventas generadas por cada uno de los empleados y el segundo, con las ventas agrupadas por las diferentes categorías.

Es obvio que las características que tienen informes y subinformes son muy similares, ya que al fin y al cabo en ambos casos se recoge información de un origen para posteriormente exponerla en un documento con un formato determinado.

Combinar informes con subinformes es un proceso clave para garantizar un informe claro, robusto y eficaz, que facilite al usuario el análisis de los datos. Sin embargo, entre ambos existen algunas diferencias que conviene tener en cuenta. Estas diferencias se recogen en el cuadro 9.3:

Cuadro 5.3 - Diferencias entre informes y subinformes

Informe	Subinformes
Puede contener subinformes	No puede contener más subinformes
Tiene encabezado y pie de página	No tiene encabezados ni pie de página
Existe como objeto independiente y principal	No tiene existencia por sí solo

5.8. Imágenes y gráficos en un informe

Incorporar diferentes elementos visuales siempre aporta un valor extra a cualquier documentación, sea del tipo que sea. A través de los mismos se consigue facilitar el análisis y comprensión de los datos, a la vez que se hace más atractiva su consulta. Lo mismo ocurre en este caso con la elaboración de los informes, de hecho, es un campo especialmente relevante en la realización de los mismos.

Evidentemente, no basta con seleccionar cualquier tipo de imagen o gráfico e incluirlo en el informe, sino que en sí mismo el elemento que se incluya debe aportar valor añadido al texto.

5.8.1. Inclusión de imágenes mediante JasperReports

En la herramienta JasperReports se proporcionan diversas opciones para incluir imágenes a partir de la paleta de elementos de la aplicación. Estas opciones se pueden comprobar en la siguiente imagen:

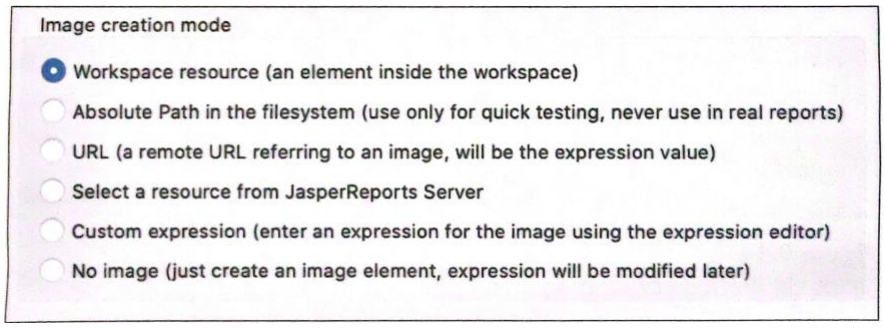


Figura 5.10 - Opciones de inclusión de imágenes en JasperReports

A las opciones más habituales para incluir una imagen (a partir de la zona de trabajo del proyecto, a partir de la ruta o de una URL) se añade la opción de diseñar una imagen desde cero, utilizando el editor de expresiones que incluye la propia herramienta.

5.8.2. Inclusión de gráficos

La inclusión de gráficos en un informe debe ser un hecho cotidiano y habitual, siempre que se trabaje con datos cuantificables. Aportan mucha claridad al informe, ya que permiten mostrar visualmente los mismos datos que puede contener el informe en forma de datos o cifras.

El tipo de gráficos que se pueden incluir en un informe es muy variado, y dependerá del tipo de información que se desee proporcionar, así como del tipo de datos. Por ejemplo, la herramienta iReport permite incluir gráficos de diferente tipo: de barras, de líneas, circulares.

Las características de estos son las siguientes:

Gráficos de barras: en los que la información se representa con barras verticales u horizontales que muestran los datos agrupados.

Gráficos lineales: este tipo de representaciones muestran valores en los ejes X e Y que aparecen unidos linealmente. Son muy útiles para representaciones temporales y en ocasiones se superponen línea para realizar comparativas, por ejemplo, el número de ventas entre departamentos a lo largo de un año.

Gráficos circulares: este último tipo se suele utilizar para representar distribuciones, ya que divide el total en diferentes fragmentos representados de la distribución.

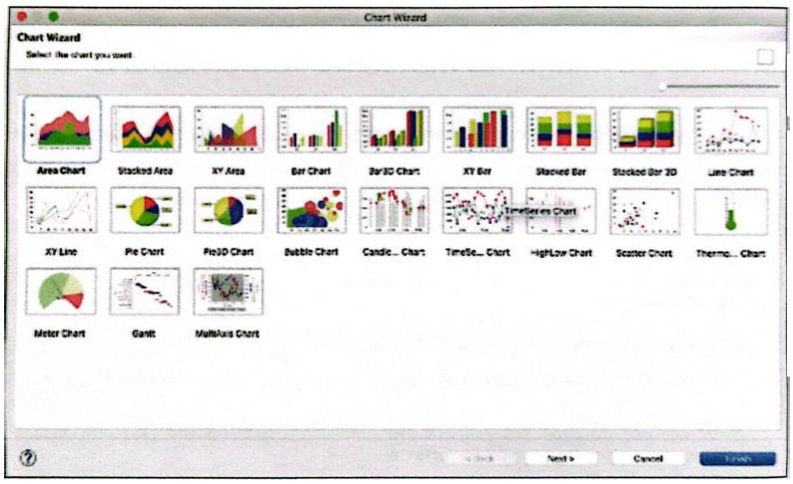


Figura 9.11 - Tipos de gráficos existentes en iReport.

Una vez generado el informe, se podrá personalizar el mismo, modificando colores, incluyendo el nombre de los ejes, mostrando líneas o cuadrículas. Conviene, en cualquier caso, incluir una leyenda clara y concisa, que ayude a comprender el contenido del mismo.