

# Desarrollo de interfaces

Diana García-Miguel López

Beatriz García-Miguel López

Agustín Fernández Herrera



**ASESOR EDITORIAL:**

---

Juan Carlos Moreno Pérez

© Diana García-Miguel López  
Beatriz García-Miguel López  
Agustín Fernández Herrera

© EDITORIAL SÍNTESIS, S. A.  
Vallehermoso, 34. 28015 Madrid  
Teléfono 91 593 20 98  
<http://www.sintesis.com>

ISBN: 978-84-135760-4-6

Impreso en España - Printed in Spain

Reservados todos los derechos. Está prohibido, bajo las sanciones penales y el resarcimiento civil previstos en las leyes, reproducir, registrar o transmitir esta publicación, íntegra o parcialmente, por cualquier sistema de recuperación y por cualquier medio, sea mecánico, electrónico, magnético, electroóptico, por fotocopia o por cualquier otro, sin la autorización previa por escrito de Editorial Síntesis, S. A.

<b>Glosario</b>	33
<b>2.1. Programación orientada a objetos</b>	33
2.1.1. Clases	33
2.1.2. Propiedades o atributos	34
2.1.3. Métodos	34
<b>2.2. Programación de eventos</b>	35
<b>2.3. Librerías de componentes</b>	36
2.3.1. AWT	36
2.3.2. Swing	37
<b>2.4. Herramientas de edición de interfaces</b>	37
<b>2.5. Contenedores</b>	38
2.5.1. Layout manager	38
2.5.2. FlowLayout	38
2.5.3. GridLayout	39
2.5.4. BorderLayout	39
2.5.5. GridBagLayout	40
<b>2.6. Componentes</b>	40
2.6.1. JButton	40
2.6.2. JLabel	41
2.6.3. JTextField	41
2.6.4. JCheckBox	42
2.6.5. JRadioButton	42
2.6.6. JComboBox	43
<b>2.7. Diálogos</b>	43
<b>Resumen</b>	44
<b>Ejercicios propuestos</b>	45
<b>Actividades de autoevaluación</b>	46
<b>3. CREACIÓN DE COMPONENTES VISUALES</b>	49
<b>Objetivos</b>	49
<b>Mapa conceptual</b>	50
<b>Glosario</b>	51
<b>3.1. Componentes visuales</b>	51
3.1.1. Concepto de componente	51
3.1.2. Propiedades y atributos	52
<b>3.2. Eventos</b>	54
3.2.1. Componentes y eventos	54
3.2.2. Listeners	55
3.2.3. Métodos y eventos	57
<b>3.3. Introspección y reflexión</b>	58
<b>3.4. Persistencia del componente</b>	58
<b>3.5. Herramientas para el desarrollo de componentes visuales</b>	59
<b>3.6. Empaquetado de componentes</b>	60
<b>Resumen</b>	61
<b>Ejercicios propuestos</b>	61
<b>Actividades de autoevaluación</b>	62

<b>4. DISTRIBUCIÓN DE APLICACIONES</b>	65
Objetivos .....	65
Mapa conceptual .....	66
Glosario .....	67
4.1. Componentes de una aplicación y empaquetado .....	67
4.2. Instaladores y paquetes autoinstalables .....	68
4.2.1. Windows .....	68
4.2.2. Linux .....	69
4.3. Interacción con el usuario .....	70
4.4. Ficheros firmados digitalmente .....	70
4.4.1. JarSigner .....	71
4.5. Instalación de aplicaciones desde un servidor web .....	72
4.6. Creación de un asistente de instalación (instalador) .....	73
Resumen .....	74
Ejercicios propuestos .....	75
Actividades de autoevaluación .....	75
<b>5. GENERACIÓN DE INTERFACES A PARTIR DE DOCUMENTOS XML</b>	77
Objetivos .....	77
Mapa conceptual .....	78
Glosario .....	79
5.1. Lenguajes de descripción de interfaces basados en XML .....	79
5.1.1. XHTML .....	80
5.1.2. GML .....	81
5.1.3. MathML .....	81
5.1.4. RSS .....	82
5.1.5. XSLT .....	82
5.1.6. SVG .....	82
5.2. El documento XML. Análisis y edición .....	84
5.2.1. Etiquetas .....	85
5.2.2. Atributos .....	85
5.2.3. Valores .....	85
5.2.4. Eventos .....	86
5.3. Herramientas libres y propietarias para la creación de interfaces de usuario multiplataforma .....	87
5.4. Generación de código para diferentes plataformas .....	88
Resumen .....	90
Ejercicios propuestos .....	91
Actividades de autoevaluación .....	92
<b>6. IMÁGENES Y SOFTWARE DE GESTIÓN DE RECURSOS GRÁFICOS</b>	95
Objetivos .....	95
Mapa conceptual .....	96
Glosario .....	97
6.1. Imágenes .....	97
6.1.1. Tipos de imagen .....	98
6.1.2. Formatos de imagen .....	99

6.1.3. Resolución y profundidad de color .....	101
6.1.4. Tamaño y compresión de imágenes .....	102
<b>6.2. Software para la gestión de recursos gráficos .....</b>	<b>103</b>
6.2.1. Software de visualización de imágenes .....	103
6.2.2. Software de edición de imágenes .....	105
6.2.3. Software de creación de imágenes .....	106
6.2.4. Software: logos e iconos .....	106
<b>6.3. Las imágenes y la ley de propiedad intelectual .....</b>	<b>107</b>
6.3.1. Derechos de la propiedad intelectual .....	107
6.3.2. Derechos de autor .....	108
6.3.3. Licencias .....	108
6.3.4. Registro de contenido .....	110
<b>Resumen .....</b>	<b>111</b>
<b>Ejercicios propuestos .....</b>	<b>112</b>
<b>Actividades de autoevaluación .....</b>	<b>112</b>
 <b>7. USABILIDAD, PAUTAS DE DISEÑO Y ACCESIBILIDAD .....</b>	<b>115</b>
<b>Objetivos .....</b>	<b>115</b>
<b>Mapa conceptual .....</b>	<b>116</b>
<b>Glosario .....</b>	<b>117</b>
<b>7.1. Usabilidad .....</b>	<b>117</b>
7.1.1. Objetivos de uso y estándares de usabilidad .....	118
7.1.2. Los usuarios .....	121
<b>7.2. Pautas de diseño. Estructura de la interfaz de una aplicación .....</b>	<b>122</b>
7.2.1. Pautas de diseño para menús .....	122
7.2.2. Pautas de diseño para ventanas y cuadros de diálogo .....	122
7.2.3. Pautas de diseño relativas al aspecto .....	123
7.2.4. Pautas de diseño para elementos interactivos .....	124
7.2.5. Pautas de diseño para la presentación de datos .....	125
<b>7.3. Accesibilidad .....</b>	<b>126</b>
7.3.1. El consorcio World Wide Web (W3C) .....	126
7.3.2. Tipos de discapacidad .....	128
<b>7.4. Análisis y verificación de la usabilidad .....</b>	<b>131</b>
7.4.1. Método por inspección. Evaluación heurística .....	131
7.4.2. Método de test con usuarios .....	132
<b>7.5. Análisis y verificación del proceso de desarrollo de interfaces .....</b>	<b>132</b>
7.5.1. Fase de planificación .....	133
7.5.2. Fase de diseño .....	133
7.5.3. Fase de implementación .....	134
7.5.4. Fase de evaluación .....	134
7.5.5. Fase de puesta en producción .....	134
7.5.6. Fase mantenimiento y seguimiento .....	134
<b>Resumen .....</b>	<b>135</b>
<b>Ejercicios propuestos .....</b>	<b>136</b>
<b>Actividades de autoevaluación .....</b>	<b>137</b>
 <b>8. DOCUMENTACIÓN DE APLICACIONES .....</b>	<b>139</b>
<b>Objetivos .....</b>	<b>139</b>
<b>Mapa conceptual .....</b>	<b>140</b>

<b>Glosario</b>	141
8.1. ¿Por qué es importante documentar?	141
8.2. Tipos de documentación	142
8.2.1. Documentación de pruebas	142
8.2.2. Documentación técnica	143
8.3. Manuales y guías	143
8.3.1. Manual y guía de usuario	144
8.3.2. Manual y guía de explotación	145
8.3.3. Guía rápida y guía de referencia	145
8.4. Ficheros de ayuda. Formatos. Herramientas para generarlos	146
8.4.1. Formatos	146
8.4.2. Herramientas para generar ficheros de ayuda	147
8.4.3. Generación de un sistema de ayuda con JavaHelp	147
8.4.4. Ayuda genérica y sensible al contexto	152
8.5. Tablas de contenidos	153
<b>Resumen</b>	154
<b>Ejercicios propuestos</b>	155
<b>Actividades de autoevaluación</b>	155
 <b>9. ELABORACIÓN DE INFORMES</b>	159
<b>Objetivos</b>	159
<b>Mapa conceptual</b>	160
<b>Glosario</b>	161
9.1. Informes en la aplicación	161
9.1.1. Informes incrustados	162
9.1.2. Informes no incrustados	162
9.2. Herramientas gráficas	163
9.3. Estructura general de un informe. Secciones, encabezados y pies	164
9.4. Formatos de salida	165
9.5. Valores calculados	166
9.5.1. Numeración de líneas. Recuentos y totales	167
9.6. Filtrado de datos. Conexión a bases de datos y diseño de consultas	168
9.6.1. Diseño de consultas	168
9.7. Subinformes	169
9.8. Imágenes y gráficos en un informe	170
9.8.1. Inclusión de imágenes mediante JasperReports	171
9.8.2. Inclusión de gráficos	171
<b>Resumen</b>	172
<b>Ejercicios propuestos</b>	173
<b>Actividades de autoevaluación</b>	174
 <b>10. REALIZACIÓN Y SISTEMAS DE PRUEBAS EN EL DESARROLLO DE INTERFACES</b>	177
<b>Objetivos</b>	177
<b>Mapa conceptual</b>	178
<b>Glosario</b>	179
10.1. El proyecto de desarrollo	179
10.1.1. Fases de un proyecto de desarrollo	179
10.1.2. Objetivos principales del sistema de pruebas	180

10.1.3. Pruebas de caja negra y caja blanca .....	180
10.1.4. Depuración de código .....	181
<b>10.2. Tipos de pruebas .....</b>	<b>182</b>
10.2.1. Pruebas unitarias .....	182
10.2.2. Pruebas de integración .....	182
10.2.3. Pruebas de regresión .....	183
10.2.4. Pruebas funcionales .....	183
10.2.5. Pruebas no funcionales: de capacidad, rendimiento, uso de recursos y seguridad .....	184
10.2.6. Pruebas manuales .....	185
10.2.7. Pruebas automáticas .....	185
10.2.8. Pruebas de usuario .....	186
10.2.9. Pruebas de aceptación .....	186
10.2.10. Desarrollo del plan de pruebas .....	187
<b>10.3. Versiones alfa y beta .....</b>	<b>187</b>
10.3.1. Versión ALFA .....	188
10.3.2. Versión BETA .....	188
<b>Resumen .....</b>	<b>189</b>
<b>Ejercicios propuestos .....</b>	<b>189</b>
<b>Actividades de autoevaluación .....</b>	<b>190</b>

# Presentación

En la actualidad vivimos en un mundo en constante evolución, donde el uso de aplicaciones para casi cualquier ámbito está cada vez más presente. Es por ello que se hace imprescindible que la combinación entre información e interacción con los usuarios sea un aspecto clave en el desarrollo de aplicaciones. Ya no basta con crear aplicaciones sin más, sino que debe proporcionarse a los usuarios una experiencia de uso satisfactoria.

Para garantizar el éxito de una aplicación es clave proporcionar un acceso rápido y óptimo a todas las funcionalidades de la herramienta, así como crear una experiencia visual agradable al usuario, que lo invite a volver a utilizarla en el futuro. La consecución o no de los fines de una determinada aplicación dependerán, en gran medida, de la satisfacción que se proporcione al usuario. Por tanto, es fundamental un diseño adecuado de la aplicación; para garantizar este objetivo, trabajaremos sobre la usabilidad y la accesibilidad de las interfaces, entre otros aspectos importantes.

El libro que tienes entre tus manos se centra en el desarrollo de interfaces y componentes visuales, basándose principalmente en el lenguaje de programación en Java y las librerías asociadas, así como el uso de XML para el desarrollo de interfaces. El presente libro va dirigido a todo aquel que quiera comenzar a investigar y formarse en el desarrollo de interfaces para aplicaciones, tan importante en la actualidad. En concreto, se dirige al alumnado del módulo Desarrollo de Interfaces del ciclo formativo de grado superior de Desarrollo de Aplicaciones Multiplataforma.

La estructura de este libro se encuentra dividida en tres bloques temáticos claramente diferenciados. Por un lado, en los primeros capítulos se realiza un primer acercamiento a la importancia del desarrollo de interfaces, el lenguaje de desarrollo de componentes visuales, el uso de entornos de desarrollo (IDE) y la creación de componentes visuales.

El segundo bloque se encuentra constituido por los capítulos relativos a la usabilidad y la selección adecuada de las imágenes y otros elementos multimedia que son claves para el de-

sarrollo una adecuada interfaz. Finalmente, el tercer bloque recoge ampliamente los capítulos relativos a la generación de informes, documentaciones y manuales.

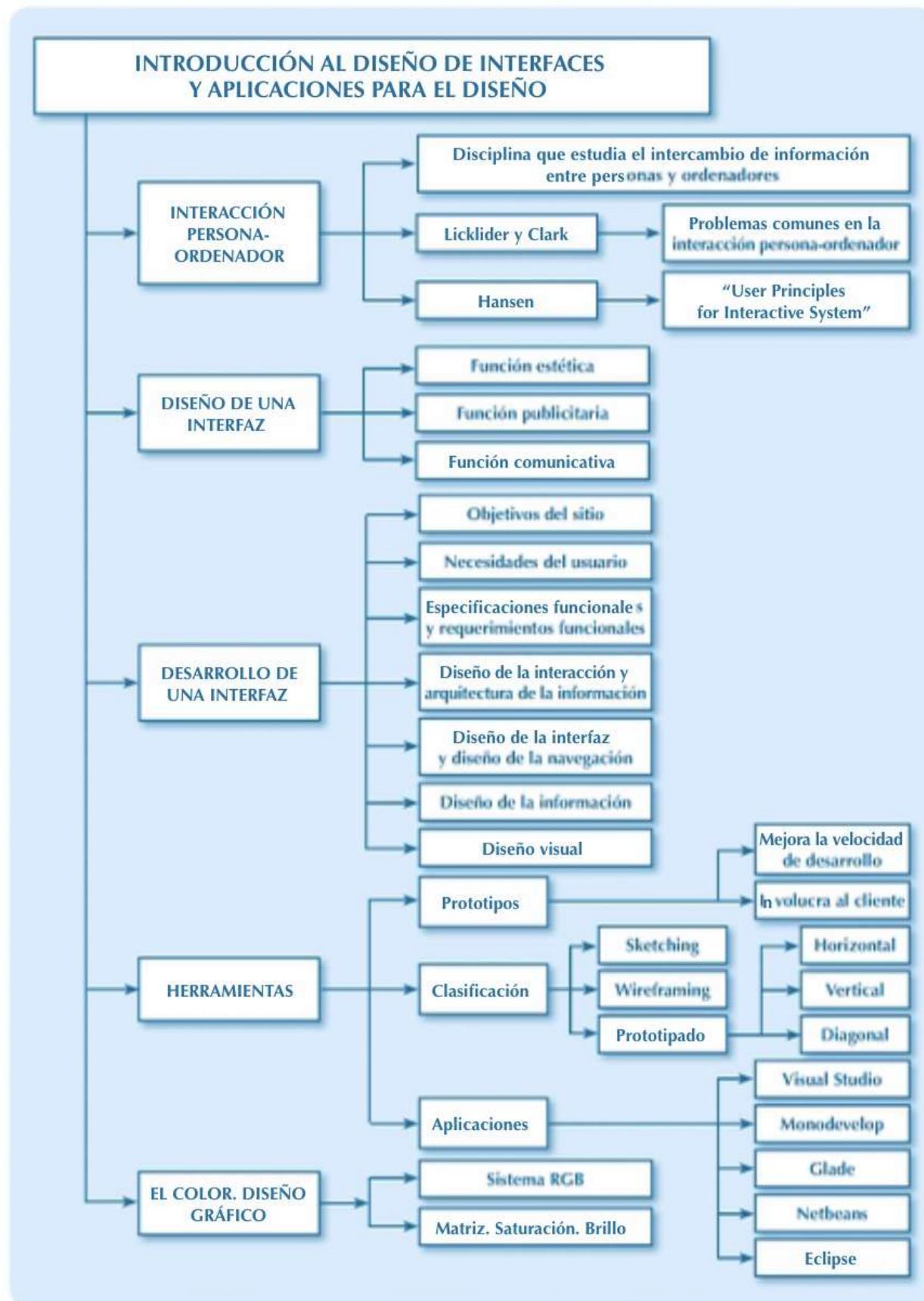
En este libro también podrán encontrarse algunas herramientas que están a nuestro alcance y se explicará el paso a paso para su correcto uso. La estructura interna de cada uno de los capítulos está formada por el contenido propio de la explicación de los apartados, acompañado en todo momento por reglas y resúmenes de sintaxis y múltiples ejemplos de código. El contenido de este libro y el estudio de este módulo pueden ser el comienzo de un prometedor futuro como diseñadores de interfaces.

# Introducción al diseño de interfaces y aplicaciones para el diseño

## Objetivos

- ✓ Reconocer la importancia de la comunicación visual y sus principios básicos.
- ✓ Identificar y analizar los elementos para la elaboración de prototipos.
- ✓ Planificar el proceso de elaboración del diseño de una interfaz.
- ✓ Analizar los distintos entornos de desarrollo y escoger el que más se adecua al diseño de cada proyecto.

## Mapa conceptual



## Glosario

**API (*Application Programming Interfaces*)**. Interfaz de programación de aplicaciones.

**Diseño gráfico**. Acción de programar, proyectar y realizar comunicaciones visuales de aplicaciones u otras herramientas software que generalmente van a ser transmitidas por medios industriales.

**Eclipse**. Entorno de desarrollo de código abierto y multiplataforma. Dispone de la funcionalidad Graphical Layout que nos permite visualizar el contenido en vista de diseño y desarrollar componentes visuales de una forma rápida e intuitiva.

**IDE (*Integrated Development Environment*)**. Entorno de desarrollo integrado. Aplicación informática que proporciona servicios para facilitar el desarrollo de software.

**Interacción**. Disciplina que estudia el intercambio de información entre las personas y los ordenadores, cuyo objetivo es que este intercambio sea más eficiente y se incremente la satisfacción.

**NetBeans**. Herramienta de código abierto. Se trata de uno de los entornos de desarrollo más utilizados en cuanto al desarrollo de interfaces a través de lenguaje de programación en Java.

**Prototipo**. Consiste en una maqueta o modelo de un diseño o dispositivo para que nos hagamos una idea del producto final que se obtendrá.

**Sketching**. Prototipo en el que se dibuja toda la interfaz de la aplicación, junto con los procesos y las relaciones entre pantallas. Se suele emplear en la fase inicial y se basa en el establecimiento de la jerarquía de contenidos, pero sin detalles de diseño.

**Wireframing**. Para dibujar con un cierto nivel de detalle las pantallas, sus esbozos de contenido, las llamadas a la acción y, en general, la disposición física de los elementos (papel o digital).

### 1.1. Interacción persona-ordenador

Lo primero que debemos plantearnos a la hora de diseñar una interfaz es *¿qué es la interacción persona-ordenador?* Podemos encontrar multitud de definiciones, pero nos vamos a quedar con que es la disciplina que estudia el intercambio de información entre las personas y los ordenadores, cuyo objetivo es que este intercambio sea más eficiente, es decir, disminuyan los errores, se incremente la satisfacción, etc.

Licklider y Clark, en 1962, elaboran una lista con los diez problemas más comunes que deberán ser resueltos para facilitar la interacción entre las personas y los ordenadores:

1. Compartir el tiempo de uso de los ordenadores entre muchos usuarios.
2. Un sistema de entrada-salida para la comunicación mediante datos simbólicos y gráficos.
3. Un sistema interactivo de proceso de las operaciones en tiempo real.

4. Sistemas para el almacenamiento masivo de información que permitan su rápida recuperación.
5. Sistemas que faciliten la cooperación entre personas en el diseño y programación de grandes sistemas.
6. Reconocimiento por parte de los ordenadores de la voz, de la escritura manual impresa y de la introducción de datos a partir de escritura manual directa.
7. Comprensión del lenguaje natural, sintáctica y semánticamente.
8. Reconocimiento de la voz de varios usuarios por el ordenador.
9. Descubrimiento, desarrollo y simplificación de una teoría de algoritmos.
10. Programación heurística o a través de principios generales.

Hansen (1971) en su libro *User Engineering Principles for Interactive Systems* hace la primera enumeración de principios para el diseño de sistemas interactivos:

1. Conocer al usuario.
2. Minimizar la memorización, sustituyendo la entrada de datos por la selección de ítems, usando nombres en lugar de números, asegurándose un comportamiento predecible y proveyendo acceso rápido a información práctica del sistema.
3. Optimizar las operaciones mediante la rápida ejecución de operaciones comunes, la consistencia de la interfaz y organizando y reorganizando la estructura de la información basándose en la observación del uso del sistema.
4. Facilitar buenos mensajes de error, crear diseños que eviten los errores más comunes, haciendo posible deshacer acciones realizadas y garantizar la integridad del sistema en caso de un fallo de software o hardware.

### Actividad propuesta 1.1



Según los problemas definidos por Licklider y Clark, ¿qué solución o soluciones podrías proponer para facilitar la interacción persona-ordenador?

## 1.2. Diseño de una interfaz. El diseño gráfico

El diseño gráfico consiste en programar, proyectar y realizar comunicaciones visuales de aplicaciones u otro tipo de herramientas software que generalmente serán transmitidos por medios industriales. A día de hoy, esta área de desarrollo tiene a sus propios profesionales, denominados *diseñadores gráficos*. Se destacan sus tres grandes funciones:

- Función estética.
- Función publicitaria.
- Función comunicativa.

Se distinguen cuatro grupos de elementos en el diseño de interfaces: elementos conceptuales, elementos visuales, elementos de relación y elementos prácticos.

Por lo tanto, la interfaz será un conjunto de elementos gráficos y un diseño de su distribución que permite una mejor presentación y navegación a través de la aplicación. Si no existen ambos factores unidos, si el resultado final de nuestro sitio es óptimo, será fruto de la casualidad. Podemos ver en las siguientes imágenes dos diseños, uno bueno y otro no tanto.



**Figura 1.1**  
Diseños de contrapuesto: un buen y un mal diseño gráfico.

Como se puede ver en la figura 1.1, aparecen contrapuestas dos interfaces para una misma aplicación, en la de la izquierda podemos observar un “mal” diseño, se trata de una aplicación en la que los elementos no aparecen claramente diferenciados, no se puede leer bien el texto, no se aprecian bien las imágenes, etc. Mientras que, en la aplicación de la derecha, la navegación del usuario resulta mucho más intuitiva, proporcionando un mayor grado de satisfacción, lo que se debe, entre otros factores, a que los elementos aparecen claramente dispuestos y no hay sobrecarga de estos.



### Actividad propuesta 1.2

Busca alguna aplicación que, al igual que en la figura 1.1, se pongan de manifiesto las diferencias entre una aplicación con un buen diseño y otra con un mal diseño. ¿Qué te hace clasificarlos dentro de esas categorías? ¿Qué partes consideras que son las más importantes?

En la construcción de cualquier aplicación y su interfaz correspondiente, debemos tener en cuenta diferentes fases, desde la definición de los objetivos que se persiguen con nuestro proyecto hasta el diseño visual resultante, pasando por las especificaciones funcionales, entre otros.

En la siguiente pila, vamos a definir todas estas fases que se deben tener presentes en el desarrollo de la interfaz.



**Figura 1.2**  
Plano de jerarquía de diseño de una aplicación.

Tras identificar las tareas que debe realizar una aplicación, en primer lugar definiremos los objetos y las acciones; este enfoque es similar a la definición de clases en un diseño orientado a objetos. Lo primero que se aconseja es realizar un modelado textual del escenario de la aplicación, distinguiendo entre sustantivos y verbos, los primeros defienden los objetos y elementos, y los segundos las acciones sobre los anteriores. Es habitual diferenciar los siguientes tipos de objetos:

- Objetos origen*: elementos de la aplicación que permiten ser desplazados a cualquier otro punto, siendo depositados sobre un objeto destino.
- Objetos destino*: elementos que reciben los objetos origen.
- Objetos de la aplicación*: elementos propios de la aplicación que no permiten una interacción directa con el usuario.

Uno de los aspectos más importantes en cuanto al diseño de interfaces recae sobre el tipo de la pantalla donde será mostrado, se ha de analizar por completo el diseño gráfico que vamos a emplear, la colocación de los iconos, el dispositivo de interacción, los textos descriptivos, etc. Una vez finalizado el diseño, debemos realizar una revisión del mismo para descubrir los posibles problemas de funcionamiento y su corrección.

- ✓ *Tiempo de respuesta del sistema*. El tiempo de respuesta hace referencia a dos conceptos clave: duración y variabilidad. El primero queda definido como el tiempo que se emplea para completar una acción. Escoger este valor no es trivial, puesto que si es muy corto el usuario puede no haber completado la acción, pero si es demasiado largo, puede suponer su desistimiento.
- ✓ *Servicios de ayuda al usuario*. Existen dos tipos de ayuda al usuario: ayudas integradas en la aplicación, que aparecen en la mayoría de casos de forma automática, como sugerencia o guía al iniciar una tarea; y ayudas complementarias, que normalmente se encuentran en forma de manuales disponibles en la propia herramienta o a través de Internet. Estos manuales permiten filtrar el contenido que se busca a través de la generación de pequeñas consultas.
- ✓ *Etiquetado de órdenes*. Este aspecto se centra en el correcto diseño de la nomenclatura asociada a cada acción, es decir, que el elemento textual o visual correspondiente a una determinada función sea una palabra adecuada para cada acción, así como incluir atajos de teclado significativos y fáciles de recordar, o emplear teclas de función para las órdenes más usadas.

## 1.3. Herramientas

El primer paso para el diseño de una interfaz es la elaboración de una maqueta o prototipo, la consecución de una buena versión previa de lo que más adelante se va a desarrollar mejora la velocidad de desarrollo de la aplicación. Existen diferentes tipos de herramientas que nos permiten llevar a cabo una adecuada construcción de prototipos, incluyendo ventanas, menús, tratamiento de errores, cuadros de diálogo, etc.

Uno de los puntos más importantes para definir un buen diseño es la evaluación del mismo, el diseño de esta puede resultar determinante. Se aconseja al menos llevar a cabo el siguiente proceso:

1. A través de un formulario de carácter cuantitativo o cualitativo se recoge el grado de satisfacción del usuario. En el segundo de los casos será posible evaluar el tiempo empleado por el usuario para aprender el funcionamiento de la aplicación, el tiempo de lectura de la ayuda, etc.
2. Gracias a los datos recogidos, el diseñador puede comprobar si la interfaz cumple los requisitos planteados al comienzo del diseño.
3. Si el análisis anterior no cumple las expectativas de diseño, será necesario revisar de nuevo el mismo para que se adecue a las directrices iniciales, así como para solventar los errores o defectos que se hubieran podido detectar. Este proceso se repetirá hasta que se cumplan los requisitos previos de diseño para la interfaz.

Las herramientas disponibles para el desarrollo de entornos gráficos pueden ordenarse en una primera clasificación entre comerciales y libres. Las primeras son las desarrolladas por empresas para su venta; dentro de esta categoría encontramos Microsoft y Borland.

### 1.3.1. Prototipos. Elementos clave de un prototipo

Un prototipo consiste en una maqueta o modelo de un diseño para que nos hagamos una idea del producto final que se obtendrá. El prototipo nos permite ver el resultado de distintos diseños finales, comprobar alguna funcionalidad o realizar test de usabilidad. De esta forma, ahorraremos tiempo, esfuerzo o dinero, puesto que es más sencillo realizar cambios sobre un diseño previo y no sobre un producto final.

Aplicando el uso de prototipos al diseño de interfaces, su implementación es fundamental, puesto que permite realizar multitud de diseños previos de los menús, elementos y demás partes del diseño para adecuarlos a las necesidades del cliente, antes de comenzar a escribir el código.

Algunas de las ventajas que justifica el uso del prototipo son:

- *Mejora la velocidad de desarrollo.* Es más eficiente realizar los cambios sobre un prototipo antes de comenzar su desarrollo que sobre un diseño definitivo, en el que aparezcan colores y tipografías, puesto que el cliente perderá la atención en el diseño base, que es el objeto principal del diseño de interfaces y del prototipo como primer paso de desarrollo.
- *Involucra al cliente.* Como se ha comentado en el punto anterior, el cliente es el responsable de aprobar el diseño último del sitio, por lo tanto, será mucho más sencillo y útil

proponer y hacer cambios sobre un esquema con poco detalle que sobre un diseño ya acabado. Es fácil involucrar al cliente en esta fase.

No existe un tipo único de esquemas de prototipo, sino que encontramos múltiples, como esquemas de página, wireframes, prototipos, mockups, bocetos, sketches, diagramas. Se pueden distinguir tres tipos, tal y como indica Daniel Torres Burriel:

- ✓ *Sketching*: para dibujar toda la interfaz de la aplicación, los procesos y las relaciones entre pantallas (solo papel). Este primer tipo es el que se suele emplear en la fase inicial, donde realizaremos diseños esquemáticos en papel. Se basa esencialmente en establecer la jerarquía de contenidos, pero sin detalles de diseño.
- ✓ *Wireframing*: se utiliza para dibujar con un cierto nivel de detalle las pantallas, los esbozos de contenido, las llamadas a la acción y, en general, la disposición física de los elementos (papel o digital). En esta fase se desarrolla y entrega una maqueta con base en lo “diseñado” en el paso previo. De esta forma es posible validar los aspectos de diseño por parte del cliente. Lo fundamental en esta fase es la organización de los contenidos.
- ✓ *Prototipado*: para diseñar y ejecutar la interacción entre las pantallas que componen los procesos (solo digital). El prototipado se utiliza como paso final, puesto que permite evaluar no solo el diseño y la organización, sino también el funcionamiento, la interacción (menús, formularios, botones, iconos). Se emplea para hacer pruebas de usuarios antes de tener hecho el desarrollo completo e implantado del producto. Esto ahorra horas de desarrollo, ya que son necesarias menos versiones de la aplicación. Podemos encontrar tres clases de prototipos basados en su funcionalidad:
  - a) *Horizontal*. Modela muchas características de una aplicación, pero incorporando pocos detalles. Es el prototipo utilizado en las primeras etapas de diseño.
  - b) *Vertical*. Modela pocas características, pero en este caso se añaden más detalles.
  - c) *Diagonal*. Se trata de un prototipo mixto entre los dos anteriores. Hasta cierto nivel presenta las características del tipo horizontal, a partir del cual implementa las del tipo vertical.

### 1.3.2. Aplicaciones para el desarrollo de interfaces

El diseño del prototipo, en cualquiera de sus fases, se debe basar en los siguientes aspectos:

- Identificar los elementos que forman parte de cada una de las pantallas de una aplicación.
- Distribuir el número de elementos de la interfaz gráfica para que no exista saturación de elementos, pero haya suficiente información en la misma y la interacción sea correcta.
- Organización de la jerarquía de elementos, orden y disposición de los mismos.
- Extensión adecuada del diseño para aprovechar eficientemente el espacio en función del dispositivo.
- Patrones de diseño para estandarizar el de interfaces.
- Aspectos técnicos de usabilidad y accesibilidad.

Cuando hablamos de una interfaz gráfica, podemos diferenciar claramente dos áreas, aquella común a todas las ventanas de una misma aplicación y, por otro lado, la parte de contenido que varía de una ventana a otra.



### Actividad propuesta 1.3

Realiza un primer prototipo en el que definirás el funcionamiento de una aplicación. A lo largo de este libro adquirirás herramientas para poder implementarla tanto gráfica como funcionalmente.



PARA SABER MÁS

La *jerarquía visual* es la disposición de los elementos. No se trata de algo trivial, sino que es necesario definir de forma eficiente las prioridades de comunicación, información e interacción.

Los elementos se deben situar de izquierda a derecha y de arriba hacia abajo para establecer una jerarquía que va de mayor a menor importancia. Lo más importante va arriba a la izquierda, y va perdiendo fuerza o importancia aquello que se relega hacia abajo y a la derecha.

El uso de herramientas basadas en componentes visuales para el desarrollo de interfaces presenta numerosas ventajas, pero una de las más importantes es la facilidad con la que se pueden empaquetar los componentes desarrollados para ser reutilizados posteriormente. La reutilización del código es un aspecto clave para la implementación de nuevos proyectos, puesto que simplifica el desarrollo y permite prestar atención a aspectos más importantes y no repetir fragmentos de código previamente desarrollados y probados. Algunas de las herramientas más conocidas en la actualidad se muestran a continuación:

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

Este entorno de desarrollo incorpora la funcionalidad de autocompletado de código, lo que permite detectar los problemas en tiempo real. A la hora de depurar el código, permite ir paso a paso, estableciendo puntos de interrupción, por procedimientos y por instrucciones. Por último, cabe destacar que permite administrar el código en los repositorios más utilizados en la actualidad, como son GIT, GitHub y Azure DevOps. De esta forma es posible controlar las modificaciones entre las diferentes versiones de nuestros proyectos y poder realizar copias de seguridad de los archivos durante el desarrollo del mismo.

- ✓ *Monodevelop*. Este IDE es libre y gratuito, incorpora todas las funcionalidades propias de un editor de texto además de otras que permiten depurar y gestionar proyectos. Entre sus principales ventajas se encuentran que permite trabajar con algunos de los lenguajes más demandados en la actualidad, como son C#, Java, .NET y Python. Pertenece a Unity, motor de videojuegos multiplataforma por excelencia.

**CUADRO 1.1**

## Comparativa de las herramientas de edición de interfaces

Nombre	Licencia	Lenguajes soportados	Enlace
Visual Studio *Community	Propietaria *Libre	C#, HTML, Javascript, XML	<a href="https://visualstudio.microsoft.com/es/">https://visualstudio.microsoft.com/es/</a>
Mono Develop	Libre	C#, Java, .NET, Python	<a href="https://www.monodevelop.com">https://www.monodevelop.com</a>
Glade	Libre	C++, C#, Java, Python	<a href="https://glade.gnome.org">https://glade.gnome.org</a>
NetBeans	Libre	Java, HTML, PHP, Python	<a href="https://netbeans.org">https://netbeans.org</a>
Eclipse	Libre	Java, C++, PHP	<a href="https://www.eclipse.org">https://www.eclipse.org</a>

- ✓ *Glade.* Permite la creación de interfaces gráficas de usuario. Es muy utilizada en entornos XML. Permite el desarrollo de interfaces gráficas basadas en lenguaje C, C++, C#, Java, Python.
- ✓ *NetBeans.* Herramienta de código abierto. Se trata de uno de los entornos de desarrollo más utilizados en cuanto al desarrollo de interfaces a través de lenguaje de programación en Java. Este IDE permite extender el entorno con un gran número de módulos que agrupan clases de Java que permiten interactuar con las API de NetBeans.
- ✓ *Eclipse.* Entorno de desarrollo de código abierto y multiplataforma. Dispone de la funcionalidad Graphical Layout que nos permite visualizar el contenido en vista de diseño y desarrollar componentes visuales de una forma rápida e intuitiva. Cabe destacar su componente Palette, un panel que permite crear botones, cuadros de texto, cuadrículas, insertar imágenes.



**Figura 1.3**  
Logotipos de IDE.

## 1.4. Planteamiento y diseño de una interfaz

Para finalizar este primer capítulo de introducción, en el que ya hemos visto la importancia del diseño en las interfaces, así como algunas herramientas que nos pueden resultar útiles, nos centramos en los elementos clave que debemos tener en cuenta para crear el prototipo de nuestra interfaz.

- a) Los elementos que van a formar parte de nuestra aplicación. Debe haber un número suficiente, pero sin que haya saturación de estos.
- b) La extensión de la aplicación.
- c) Patrones de diseño que van a utilizarse para estandarizar el diseño de interfaces.
- d) Aspectos técnicos de usabilidad y accesibilidad.



**Figura 1.4**  
Pasos para el diseño de un sitio web.

### 1.4.1. Área de redacción

El área de redacción es la encargada de delimitar los pilares fundamentales de los que debe constar el proyecto que se está desarrollando. El aspecto principal que se debe establecer es el objeto final que busca nuestra aplicación. Por ejemplo, no es lo mismo realizar el diseño de una aplicación que se va a dedicar a vender respuestas de neumáticos, que una aplicación para la venta de entradas de conciertos.

### 1.4.2. Área de producción

Una vez que se ha definido el propósito final del proyecto, es necesario que este se encuentre en consonancia con las circunstancias del mercado actual, llevar a cabo un estudio de la viabilidad y trazar el plan de desarrollo más adecuado. Los agentes que participan en la producción son:

- *Cliente*. Se trata de quién encarga la creación y desarrollo de la aplicación del proyecto.
- *Usuario*. El público hacia el que va dirigida la aplicación. Son los receptores finales del proyecto, por lo tanto, es interesante hacer un estudio de estos, conocer sus necesidades y demandas, con el fin de mejorar el rendimiento de la aplicación.
- *Presupuesto*. Se trata del total económico que el cliente desea destinar a la construcción de la aplicación. En base a este se escogerá el gestor de contenidos, entre otros elementos.
- *Plan de trabajo*. Calendario de entregas, distribución de las tareas, etc.

### 1.4.3. Área técnica

Esta área se encuentra constituida por los responsables técnicos, que son los encargados de realizar un estudio de los requisitos del proyecto relativos a su programación, normalmente en HTML, CSS y bases de datos. Hasta aquí el resto de las áreas se han basado en uso de *sketching*, en un primer momento, y en *wireframing*. El área técnica comienza el desarrollo de los primeros prototipos y maquetas. Recordemos que estos ya comienzan a incorporar los primeros elementos de interacción, hasta llegar al producto final.

#### 1.4.4. Área artística

Esta última área se centra en la estética final del proyecto, en base al propósito final de uso de la aplicación, así como a los informes de usuario y los requisitos del cliente. Hay que tener en cuenta que el estilo variará en función de la aplicación y de los usuarios. El estilo del proyecto se debe convertir en una seña de identidad, en esto consiste realizar un buen diseño gráfico, dotar de personalidad propia a una aplicación y que esta sea fácilmente reconocible en cualquier situación.

#### Actividad propuesta 1.4



- ¿Qué crees que ocurriría si en el diseño de una aplicación solo se llevase a cabo el primer paso, de definición del propósito final, y el último, de desarrollo e implementación?
- ¿Crees que se podría ahorrar tiempo al suprimir los dos pasos intermedios?

### 1.5. El color

El ojo humano solo es capaz de percibir los denominados *colores aditivos*; a través de la combinación de estos le es posible obtener el resto de los colores, esto son: azul (B), rojo (R) y verde (G). En este apartado analizaremos en qué consiste el sistema de representación RGB, así como las propiedades principales del color, que modifican y redefinen el sistema de color base.

#### 1.5.1. Sistema RGB

De la misma forma, un ordenador será capaz de obtener la representación de todos los colores utilizando el Sistema RGB, o lo que es lo mismo, el Sistema Red-Green-Blue. Indicando la proporción de cada uno de ellos dentro de la combinación de estos tres, dará lugar a toda la paleta de colores conocida.

Para representar cada color de forma que pueda ser traducido por el ordenador se utilizan 8 bits para codificar cada uno de los colores aditivos, es decir, se establece la proporción de cada color que va a formar parte de la combinación de tres. La escala monocromática de un color tendrá 256 ( $2^8$ ) valores.

A la hora de representar cada uno de los colores, es posible utilizar tanto el sistema de numeración decimal (0 a 255) como el hexadecimal, donde cada uno de los dígitos se codifica con 8 bits binarios que, agrupados en bloques de 4 bits, nos devuelve el valor correspondiente en hexadecimal.

El número de combinaciones de colores se calcula multiplicando el número máximo de grados en la escala monocromática de cada color,  $256 \times 256 \times 256$ , lo que nos da 16 777 216 colores.



**Figura 1.5**  
Círculo de colores aditivos en Sistema RGB y combinación de ellos.

## Ejemplo

El color amarillo estaría formado por:

Rojo = 255	1111 1111	ff
Verde = 255	1111 1111	ff
Azul = 0	0000 0000	00

En hexadecimal queda expresado por: #ffff00

### 1.5.2. Matiz, saturación y brillo

Además del grado en la escala monocromática de cada uno de los colores del sistema RGB, los colores presentan tres propiedades que permiten distinguirse unos de otros, estas son: el matiz, la saturación y el brillo. Estas propiedades nos permiten definir los colores como cromáticos, complementarios o cercanos, así como definir el contraste de color.

- ✓ *Matiz*. Atributo que permite distinguir un color de otro. Los tres matices primarios son los colores aditivos, verde, rojo y azul; el resto de colores se obtiene mezclando estos tres. El matiz permite definir dos colores como complementarios cuando está uno frente al otro en el círculo cromático.
- ✓ *Saturación*. Este atributo define la intensidad de un color. Puede relacionarse con el ancho de banda de luz que estamos visualizando, por lo tanto, queda condicionado por el nivel de gris presente en un color: cuanto mayor sea el nivel de gris, menos saturado será un color, y será menos intenso.



**Figura 1.6**  
Círculo cromático de matices.



**Figura 1.7**  
Escala de color con matiz rojo modificando sus valores de saturación.

- ✓ *Brillo*. Atributo que define la cantidad de luz de un color. Representa lo oscuro (si se le añade negro) o claro (si se le añade blanco) que es un color respecto de su patrón, es decir, respecto del color puro sin modificar el brillo. En una composición de colores en diseño gráfico, cuanto más brillante sea un color, más cerca parece estar.

**Figura 1.8**

Escala de color con matiz rojo modificando sus valores de brillo.



#### TOMA NOTA

En el diseño de interfaces gráficas, la selección adecuada de la carta de colores es muy importante, puesto que esto puede condicionar la experiencia de navegación del usuario, determinándola por completo. La opción más sencilla es escoger la *monocromía*, que consiste en elegir un solo color del círculo cromático y, a continuación, obtenemos su variedad de tonalidades, añadiendo blanco o negro.

Otra de las alternativas más utilizadas son los denominados *colores vecinos*. Estos son armónicos, que proporcionan estabilidad en el diseño de colores. Se denominan *vecinos* aquellos colores que se encuentran en un rango de 90° en el círculo cromático.

#### Actividad propuesta 1.5



Busca los valores que codifican los siguientes colores y exprésalos en decimal y en hexadecimal: magenta, purple, white y black.

#### Resumen

- La interacción persona-ordenador se centra en el estudio del intercambio de información entre las personas y los ordenadores, y su objetivo es que este intercambio sea más eficiente, es decir, disminuyan los errores, se incremente la satisfacción, etc. En el libro *User Engineering Principles for Interactive Systems* de Hansen (1971) se lleva a cabo una enumeración de principios para el diseño de sistemas interactivos que perdura hasta la actualidad:
  1. Conocer al usuario.
  2. Minimizar la memorización.
  3. Optimizar las operaciones mediante la rápida ejecución de operaciones comunes.
  4. Facilitar buenos mensajes de error, crear diseños que eviten los errores más comunes, haciendo posible deshacer acciones realizadas y garantizar la integridad del sistema en caso de un fallo de software o hardware.

- La base de una adecuada interacción está en el desarrollo de un correcto diseño gráfico. Este consiste en programar, proyectar y realizar comunicaciones visuales de aplicaciones, u otro tipo de herramientas software que generalmente van a ser transmitidas por medios industriales. Se destacan sus tres grandes funciones: función estética, función publicitaria y función comunicativa.
- Para comenzar el desarrollo de un proyecto de diseño de interfaz, en primer lugar, se realizan maquetas o modelos de diseño previos para crear una idea inicial del producto final, comprobar alguna funcionalidad, etc. Existen múltiples tipos de prototipos, aunque nos centramos en los tres más comunes: *sketching*, para dibujar toda interfaz de la aplicación, los procesos y las relaciones entre pantallas (solo papel); *wireframing*, fase en la que se desarrolla y entrega una maqueta en base en lo "diseñado" en el paso previo; y *prototipado*, que se utiliza como paso final, puesto que permite evaluar no solo el diseño y organización, sino también el funcionamiento, la interacción.
- Tras identificar las tareas que debe realizar una aplicación, primero definiremos los objetos y las acciones. Este enfoque es similar a la definición de las diferentes clases en un diseño orientado a objetos. Lo primero que se aconseja es realizar un modelo textual del escenario de la aplicación, distinguiendo entre sustantivos y verbos: los primeros defienden los objetos y elementos, y los segundos las acciones sobre los anteriores. Es habitual diferenciar los siguientes tipos de objetos:
  - *Objetos origen*: elementos de la aplicación que permiten ser desplazados a cualquier otro punto, siendo depositados sobre un objeto destino.
  - *Objetos destino*: elementos que reciben los objetos origen.
  - *Objetos de la aplicación*: elementos propios de la aplicación que no permiten una interacción directa con el usuario.
- El uso de herramientas basadas en componentes visuales para el desarrollo de interfaces presenta numerosas ventajas, pero una de las más importantes es la facilidad con la que se pueden empaquetar los componentes desarrollados para ser reutilizados posteriormente. La reutilización del código es un aspecto clave para la implementación de nuevos proyectos puesto que simplifica el desarrollo y permiten prestar atención a aspectos más importantes y no repetir fragmentos de código previamente desarrollados y probados. Algunas de las herramientas más conocidas en la actualidad son: Visual Studio, Monodevelop, Glade, NetBeans o Eclipse.
- El ojo humano solo es capaz de percibir los denominados colores aditivos; a través de la combinación de estos le es posible obtener el resto de los colores, esto son: azul (B), rojo (R) y verde (G). De la misma forma, un ordenador será capaz de obtener la representación de todos los colores utilizando el Sistema RGB, o lo que es lo mismo, el Sistema Red-Green-Blue. Indicando la proporción de cada uno de ellos dentro de la combinación de estos tres, se dará lugar a toda la paleta de colores conocida. Para representar cada color de forma que pueda ser traducido por el ordenador se utilizan 8 bits para codificar cada uno de los colores aditivos, es decir, se establece la proporción de cada color que va a formar parte de la combinación de tres. La escala monocromática de un color tendrá 256 ( $2^8$ ) valores.



## Ejercicios propuestos

1. ¿Cuáles son las funciones del diseño gráfico?
2. Gracias al desarrollo de interfaces gráficas se pueden utilizar imágenes y otros elementos de tipo visual para que el usuario interactúe con la aplicación o herramienta. Por eso es importante que antes de realizar la implementación del código de un programa se realice el diseño del prototipo de la herramienta donde queden recogidas las especificaciones exactas del desarrollo.

La primera interfaz que se propone desarrollar es una calculadora para ordenador. Su diseño gráfico tiene que contener las operaciones de sumar, restar, multiplicar y dividir.

  - a) ¿Qué tipo de componentes gráficos será necesario implementar en la interfaz?
  - b) ¿Cuál crees que sería el lenguaje más apropiado para desarrollar esta interfaz?
  - c) ¿Qué entorno de desarrollo elegirías?
  - d) ¿Conoces alguna librería específica para el desarrollo gráfico de interfaces?
3. Navega en los diferentes entornos de desarrollo propuestos y escoge uno de ellos para utilizarlo a lo largo de este libro. Tras completar el proceso de descarga, instala y ejecuta el programa. A continuación, crea un primer programa, por ejemplo, para que imprima por pantalla "Hola Mundo".

## ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿Qué herramienta de edición de interfaces es de tipo propietaria?:
  - a) MonoDevelop.
  - b) Glade.
  - c) NetBeans.
  - d) Visual Studio.
2. Una plantilla de diseño es:
  - a) Un lienzo en blanco sobre el que desarrollar un nuevo diseño desde cero.
  - b) Un documento de tipo wireframe.
  - c) Una plantilla prediseñada, donde el usuario solo necesita completar con el contenido deseado, y puede elegir ciertos parámetros de diseño (color, número de bloques).
  - d) Una aplicación prediseñada, donde el usuario solo necesita completar con el contenido deseado.

3. Cuando hablamos de prototipos visuales de baja calidad, un esqueleto de una aplicación, sin implementaciones de interacción, hablamos de:
- a) Prototipo.
  - b) Wireframe.
  - c) Ficheros de código HTML.
  - d) Dreamweaver.
4. ¿En qué áreas se aplica fundamentalmente el diseño?:
- a) Área de redacción y área de producción.
  - b) Área de redacción.
  - c) Área técnica y área artística.
  - d) Las respuestas a) y b) son correctas.
5. ¿Qué nombre reciben los elementos de la aplicación que permiten ser desplazados a cualquier otro punto, siendo depositados sobre un objeto destino?:
- a) Objetos origen.
  - b) Objetos destino.
  - c) Objetos base.
  - d) Objetos de la aplicación.
6. ¿Cuántos bits necesita un ordenador para representar cada color?:
- a) 2.
  - b) 4.
  - c) 8.
  - d) 6.
7. ¿Cuál de las siguientes es una ventaja del diseño de prototipos?:
- a) Mejora la velocidad de desarrollo.
  - b) Involucra al cliente.
  - c) Las respuestas a) y b) son dos de las ventajas principales.
  - d) Ninguna respuesta es correcta.
8. Señala la propiedad o propiedades de los colores que nos permiten distinguir uno de otro:
- a) Matiz y saturación.
  - b) Saturación, matiz y profundidad.
  - c) Saturación, brillo y matiz.
  - d) Ninguna de las respuestas es correcta.
9. Las tres grandes funciones del diseño gráfico son:
- a) Estética, publicitaria y comercial.
  - b) Comercial, publicitaria y estética.
  - c) Estética, publicitaria y comunicativa.
  - d) Comunicativa, estética y comercial.

10. Selecciona cuál de las siguientes opciones supone la base del diseño de prototipos:
- a) Organización de la jerarquía de elementos, orden y disposición de los mismos.
  - b) Extensión adecuada del diseño para aprovechar eficientemente el espacio en función del dispositivo.
  - c) Patrones de diseño para estandarizar el diseño de interfaces.
  - d) Todas las respuestas son correctas.

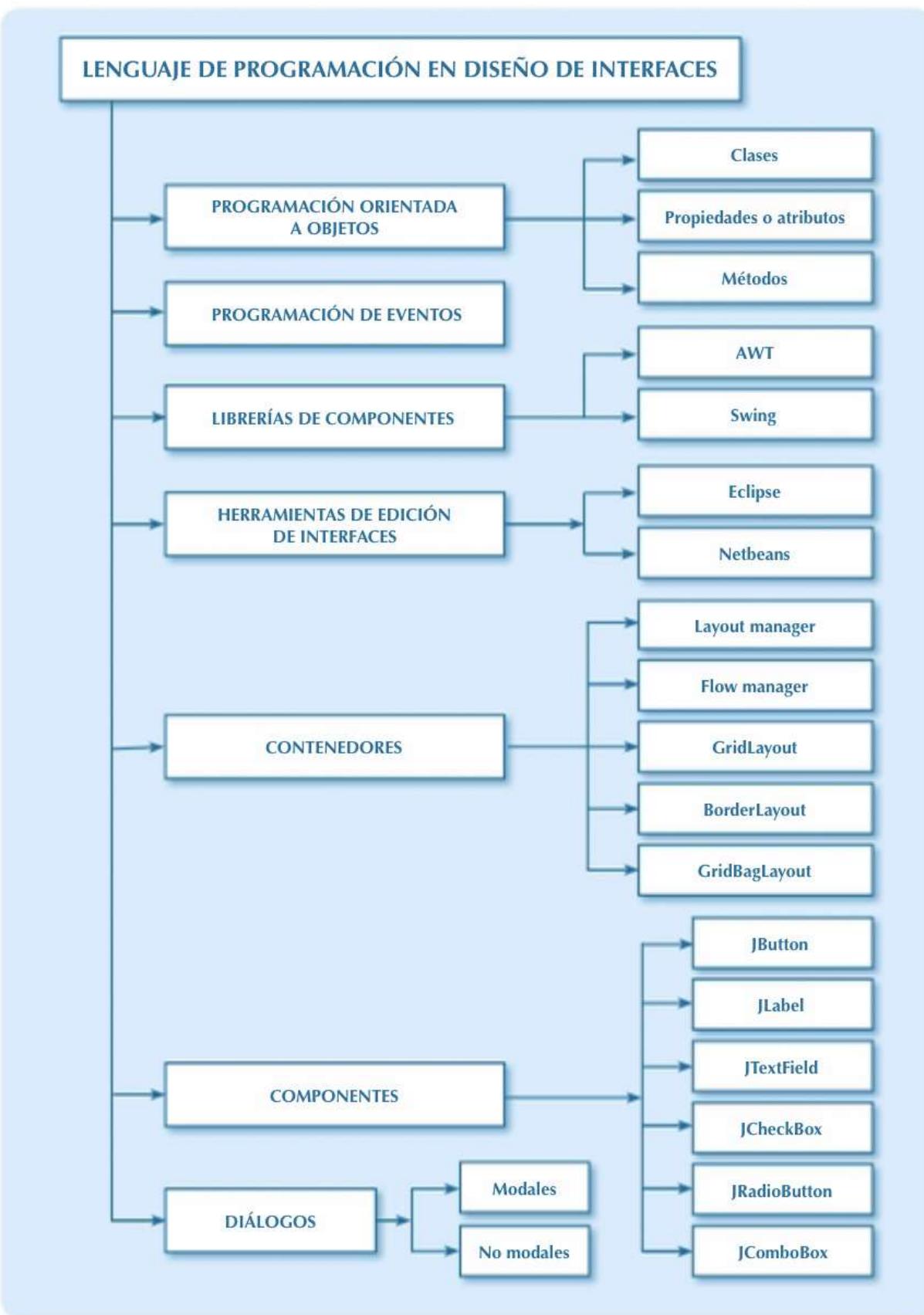
**SOLUCIONES:**1.    2.    3.    4.    5.    6.    7.    8.    9.    10.    

# Lenguaje de programación en diseño de interfaces

## Objetivos

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

## Mapa conceptual



## Glosario

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

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

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

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

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

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

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

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

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

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

## 2.1. Programación orientada a objetos

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

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

Una misma aplicación puede presentar más de una ventana, en función de la finalidad de la misma encontramos JFrame y JDialog. La segunda establece los llamados diálogos modales o no modales, elementos clave en el desarrollo de interfaces. La combinación de tipos de ventanas y elementos de diseño es infinita.

### 2.1.1. Clases

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

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

#### RECUERDA

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

### 2.1.2. Propiedades o atributos

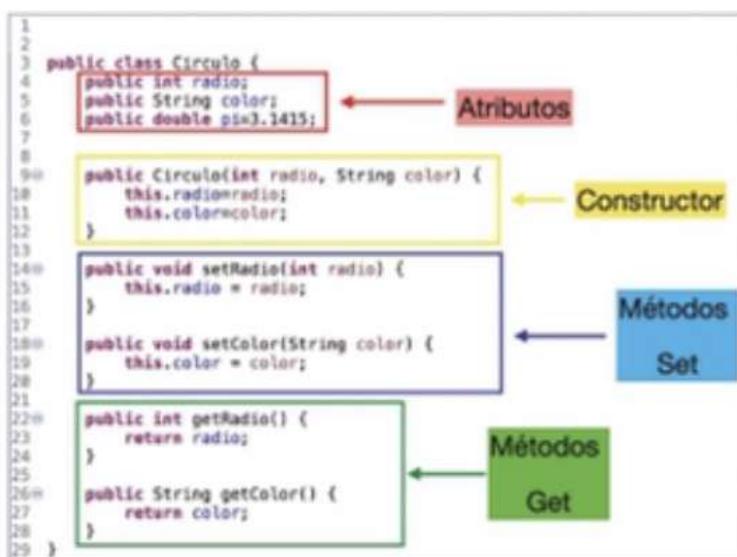
Un objeto es una cápsula que contiene todos los datos y métodos ligados a él. La información contenida en el objeto será accesible solo a través de la ejecución de los métodos adecuados, creándose una interfaz para la comunicación con el mundo exterior.

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

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

### 2.1.3. Métodos

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



**Figura 2.1**  
Estructura básica  
clase Java.

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



### Actividad propuesta 2.1

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

## 2.2. Programación de eventos

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

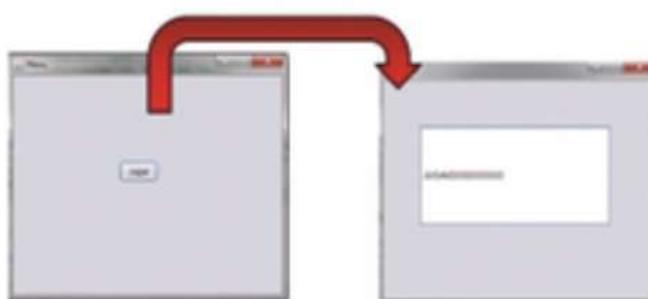
Tras la creación de las ventanas se sitúan los botones de conexión y se modifican sus propiedades de apariencia. Este elemento puede situarse dentro de un layout o de un JPanel. Para crear el evento escuchador asociado a este botón basta con hacer doble click sobre él y de forma automática se generará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

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

**Figura 2.2**  
Evento implementado para conexión de JButton.

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

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



```
private void jugarActionPerformed(java.awt.event.ActionEvent evt) {
    JUEGO J=new JUEGO();
    J.setVisible(true);
    dispose();
}
public JUEGO() {
    initComponents();
    setLocationRelativeTo(null);
    setResizable(false);
    setTitle("Juego");
}
```

**Figura 2.3**  
Conexión de ventanas con ActionListener.



## TOMA 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 propuesta 2.2



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

## 2.3. Librerías de componentes

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

### 2.3.1. AWT

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

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

**Figura 2.4**

Código para importar librerías Swing y AWT.

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

### 2.3.2. Swing

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

	AWT	SWING
Usa componentes del S.O.	✓	✗
Obliga sus propios componentes	✗	✓
El S.O maneja los eventos	✓	✗
Java maneja los eventos	✗	✓
La apariencia cambia con el S.O.	✓	✗
Tienen la misma apariencia en cualquier S.O	✗	✓
La apariencia es estática	✓	✗
Se pueden personalizar	✗	✓

**Figura 2.5**  
Comparativa de la librería AWT y Swing.



#### Actividad propuesta 2.3

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

### 2.4. Herramientas de edición de interfaces

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

WWW

**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 adapta 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 ícono que normalmente podemos identificar bajo el nombre *Eclipse Installer*.

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

## 2.5. Contenedores

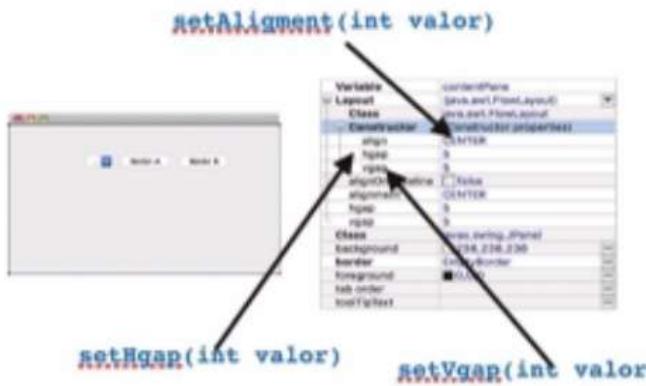
El uso de contenedores permite implementar un tipo de componente que puede contener otros componentes. Esto resulta especialmente útil para el diseño de interfaces, puesto que permite determinar la distribución y posición exacta de cada uno de sus elementos.

### 2.5.1. Layout manager

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

### 2.5.2. FlowLayout

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



**Figura 2.6**  
Propiedades FlowLayout.

### 2.5.3. GridLayout

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

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

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

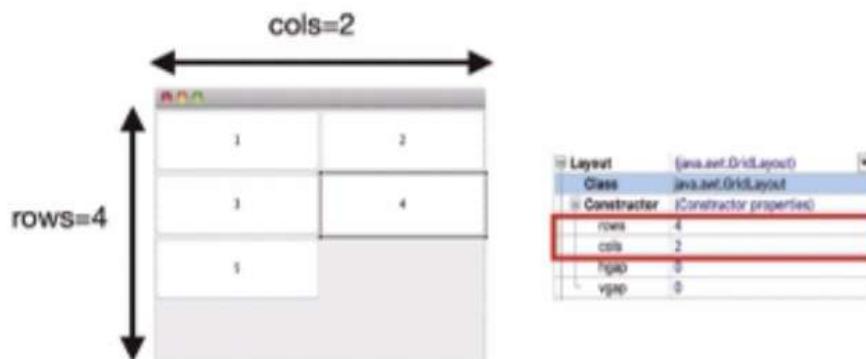


Figura 2.7

Propiedades y ejemplo de botones en GridLayout.

### 2.5.4. BorderLayout

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

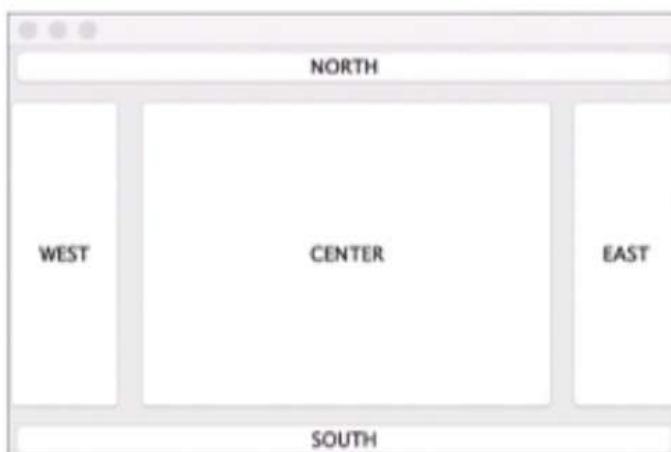


Figura 2.8

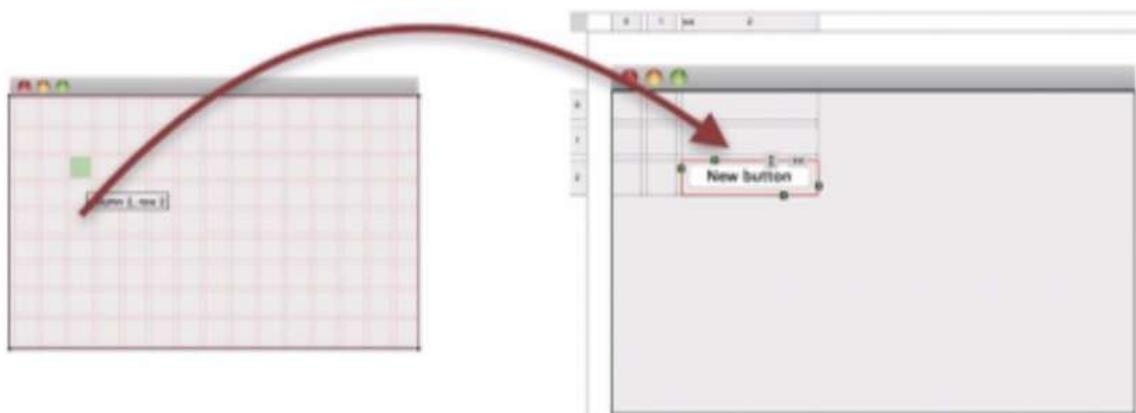
Propiedades y ejemplo de botones en BorderLayout.

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

### 2.5.5. GridLayout

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

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



**Figura 2.9**  
Ejemplo botones en GridLayout.

## 2.6. Componentes

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

### 2.6.1. JButton

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

**CUADRO 2.1**  
Propiedades JButton

background	El color de fondo del botón. Se muestra solo si es opaco.
enabled	True/false determina si el botón está activo o no.
[.../...]	

CUADRO 2.1 (CONT.)

font	Fuente del tipo de letra y tamaño.
foreground	Color del texto.
horizontalAlignment verticalAlignment	Alineación vertical y horizontal del texto con respecto al botón.
text	Texto que aparece dentro del botón.
icon	Permite cargar una imagen como fondo del botón.

## 2.6.2. JLabel

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

 CUADRO 2.2  
Propiedades JLabel

background	El color de la etiqueta si está deshabilitada.
enabled	Habilita la etiqueta.
font	Fuente del tipo de letra y tamaño.
foreground	Color del texto si la etiqueta está habilitada.
horizontalAlignment verticalAlignment	Alineación vertical y horizontal del texto con respecto a la caja de la etiqueta.
text	Texto que aparece dentro de la etiqueta.
icon	Permite cargar una imagen.

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

## 2.6.3. JTextField

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

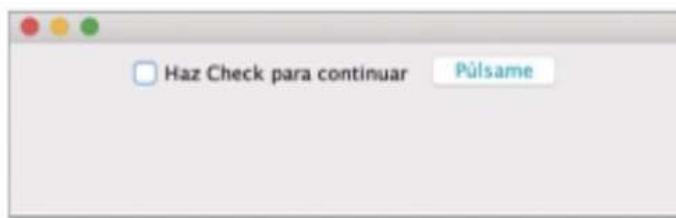
**CUADRO 2.3****Propiedades JTextField**

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

**2.6.4. JCheckBox**

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

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



**Figura 2.10**  
Interfaz ejemplo con JCheckBox y JButton.

**2.6.5. JRadioButton**

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

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

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

## 2.6.6. JComboBox

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

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

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

## 2.7. Diálogos

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

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

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

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



**Figura 2.11**  
Ventana creada JDialog.

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

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

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

**Figura 2.12**

Creación de un elemento `JDialog` de tipo modal.

### Actividades propuestas



- 2.4. ¿En qué casos crees que es más conveniente que las aplicaciones utilicen `JDialog` de tipo modal o no modal?
- 2.5. 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 herramientas de desarrollo como *Eclipse*, que permite la implementación del código o el uso de vistas de diseño para el proceso de diseño y creación de interfaces.
- El paradigma de la *programación orientada a objetos* permite manipular objetos como metáfora para simular entidades reales. Estas instancias son entidades que tienen un determinado estado, comportamiento e identidad.
- 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 botones, etiquetas, menús desplegables o casillas de verificación, entre muchos otros. Por otro lado, *Swing* es una extensión para *AWT*, un kit de herramientas de widgets utilizados para el desarrollo de interfaces gráficas en Java.
- Aunque el número de elementos que se incorporan en la paleta (*Palette*) de estas librerías es muy amplio, en este capítulo se han descrito algunos de los más usuales, analizando sus principales propiedades. El repertorio de elementos disponibles permite crear infinitas combinaciones que se adecuarán en cada caso a las especificaciones finales de la aplicación. Algunos de los elementos más importantes son:

- 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 utilizará uno u otro en función de la utilidad que se desea obtener en la interfaz.
- El análisis de eventos realizado 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.
- Lo que determinará si el diseño de una interfaz es óptimo es la elección de una buena combinación de elementos y el diseño de su distribución utilizando los elementos de tipo layout. La distribución de los elementos y formas dentro de un diseño determina la usabilidad del diseño de una aplicación.

### Ejercicios propuestos



1. ¿Qué tipo de elementos crees que pueden vincular unas ventanas con otras?
2. Para lograr la conexión entre ventanas es necesario utilizar un evento escuchador para controlar el momento en el que ocurre el evento deseado. Por ello, en este ejercicio se van a diseñar dos ventanas que se encuentran conectadas por un botón. La primera ventana es una confirmación de compra de unas entradas para una gala benéfica con botones “CANCELAR” y “PAGAR”, y la segunda ventana tiene tres opciones de modo de pago. La conexión entre ambas se llevará a cabo con el botón “PAGAR”.
  - a) ¿Qué tipo de componente será necesario implementar para crear las ventanas?
  - b) ¿Dónde será necesario asociar el evento escuchador para pagar?
  - c) ¿En qué parte de la interfaz será necesario utilizar los componentes JLabel, JRadioButton y JButton?
3. Utiliza un GridLayout para realizar el diseño de la distribución de los componentes que forman la interfaz gráfica del teclado de una calculadora. En este ejercicio se propone realizar el teclado de los números y operaciones en forma de rejilla.

## ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿Qué es una clase en Java?:

- a) Representa un conjunto de objetos que comparten una misma estructura (métodos) y comportamientos (atributos).
- b) Representa un conjunto de objetos que comparten una misma estructura (atributos) y comportamientos (métodos).
- c) Representa un objeto que hereda de otra clase.
- d) Representa un método que recibe argumentos y crea un objeto.

2. ¿Cuál de las siguientes afirmaciones respecto a los métodos de Java es cierta?:

- a) Los métodos definen la estructura de un objeto y no pueden recibir argumentos.
- b) Los métodos definen el comportamiento de un objeto y pueden recibir o no argumentos.
- c) Los métodos definen el comportamiento de un objeto y deben recibir argumentos.
- d) Los métodos definen la estructura de una clase y no pueden recibir argumentos.

3. ¿Qué es un objeto?:

- a) Las características que tienen las clases.
- b) Los métodos que tienen las clases.
- c) Una cápsula que contiene los datos y métodos ligados a él.
- d) Los métodos que reciben argumentos.

4. ¿Qué característica define a un JPanel?:

- a) Es un contenedor de componentes sobre el que se ubican los elementos.
- b) Es la ventana invisible sobre la que se sitúa un JFrame.
- c) Es un tipo de capa perteneciente a los Layouts.
- d) Es una ventana visible sobre la que se sitúa un JFrame.

5. ¿Qué diferencia existe entre JFrame y JDialog?:

- a) La ventana principal de una aplicación es un JDialog y es aconsejable utilizar para pantallas secundarias JFrame.
- b) La ventana principal de una aplicación es un JFrame y es aconsejable utilizar para pantallas secundarias JDialog.
- c) De tipo JDialog solo debe de haber una ventana.
- d) De tipo JFrame es recomendable que haya varias ventanas.

6. El componente JButton:

- a) Es el elemento de texto más utilizado.
- b) No permite utilizar imágenes.
- c) Tiene la propiedad enabled que determina si está activo o no.
- d) Se suele utilizar cuando existen varias opciones y son excluyentes.

7. Los menús desplegables se crean utilizando el componente:
- a) JComboBox.
  - b) ButtonGroup.
  - c) JTextField.
  - d) JCheckBox.
8. ¿Cuál de los siguientes manejadores de componentes tiene las propiedades “align”, “hgap” y “vgap”?:
- a) FlowLayout.
  - b) GridLayout.
  - c) BorderLayout.
  - d) GridBagLayout.
9. ¿Con qué distribución de elementos se crea por defecto un JFrame?:
- a) FlowLayout.
  - b) BorderLayout.
  - c) GridBagLayout.
  - d) GridLayout.
10. ¿Qué manejador de componentes permite un diseño más flexible y preciso?:
- a) FlowLayout.
  - b) BorderLayout.
  - c) GridBagLayout.
  - d) GridLayout.

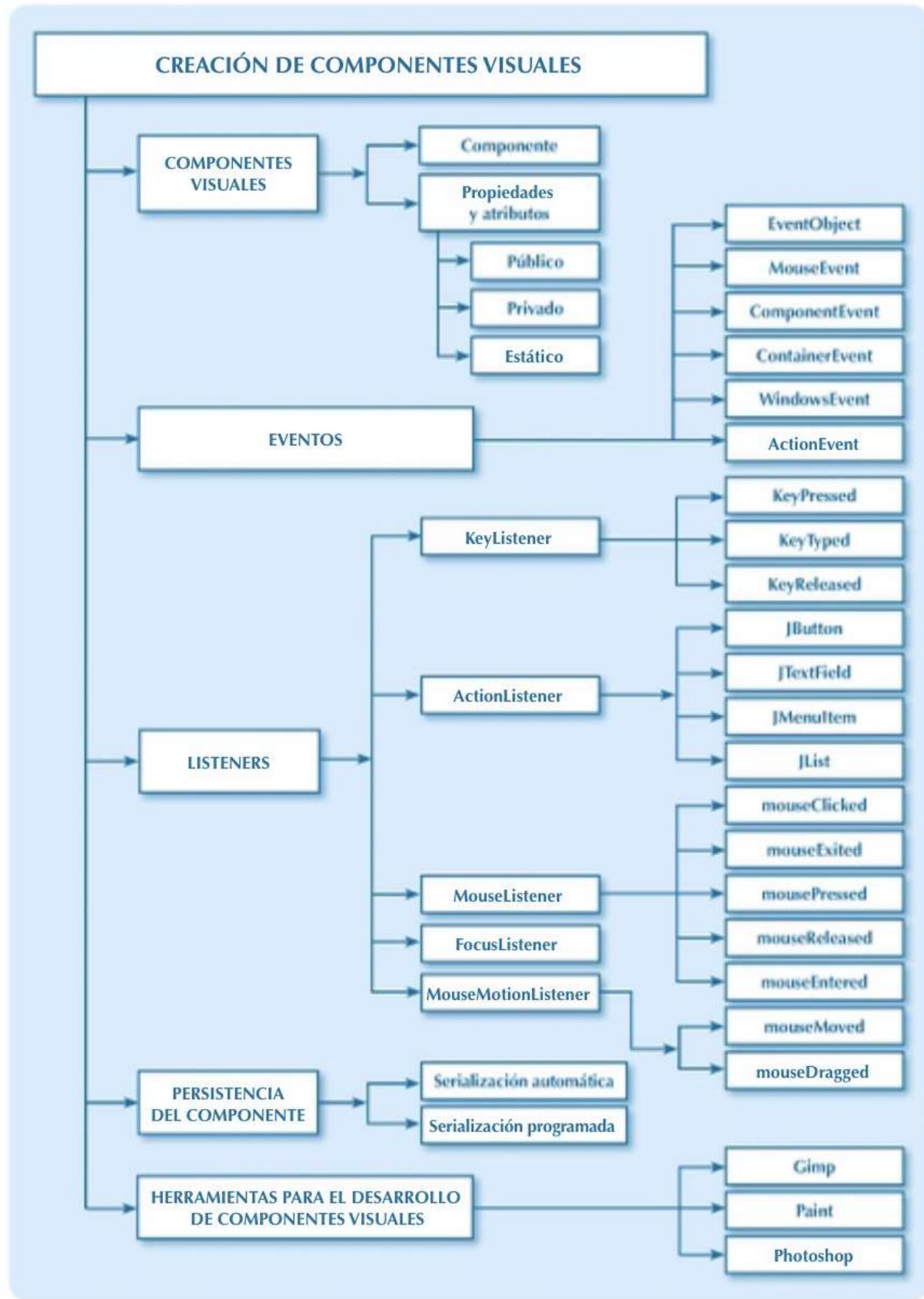
**SOLUCIONES:**1. **a** **b** **c** **d**2. **a** **b** **c** **d**3. **a** **b** **c** **d**4. **a** **b** **c** **d**5. **a** **b** **c** **d**6. **a** **b** **c** **d**7. **a** **b** **c** **d**8. **a** **b** **c** **d**9. **a** **b** **c** **d**10. **a** **b** **c** **d**

# Creación de componentes visuales

## Objetivos

- ✓ Crear componentes visuales.
- ✓ Definir propiedades de los componentes y asignar valores por defecto.
- ✓ Determinar los eventos a los que debe responder el componente y asociar las acciones correspondientes.
- ✓ Empaquetar componentes.
- ✓ Programar aplicaciones cuyo interfaz gráfico utiliza los componentes creados.

## Mapa conceptual



## 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

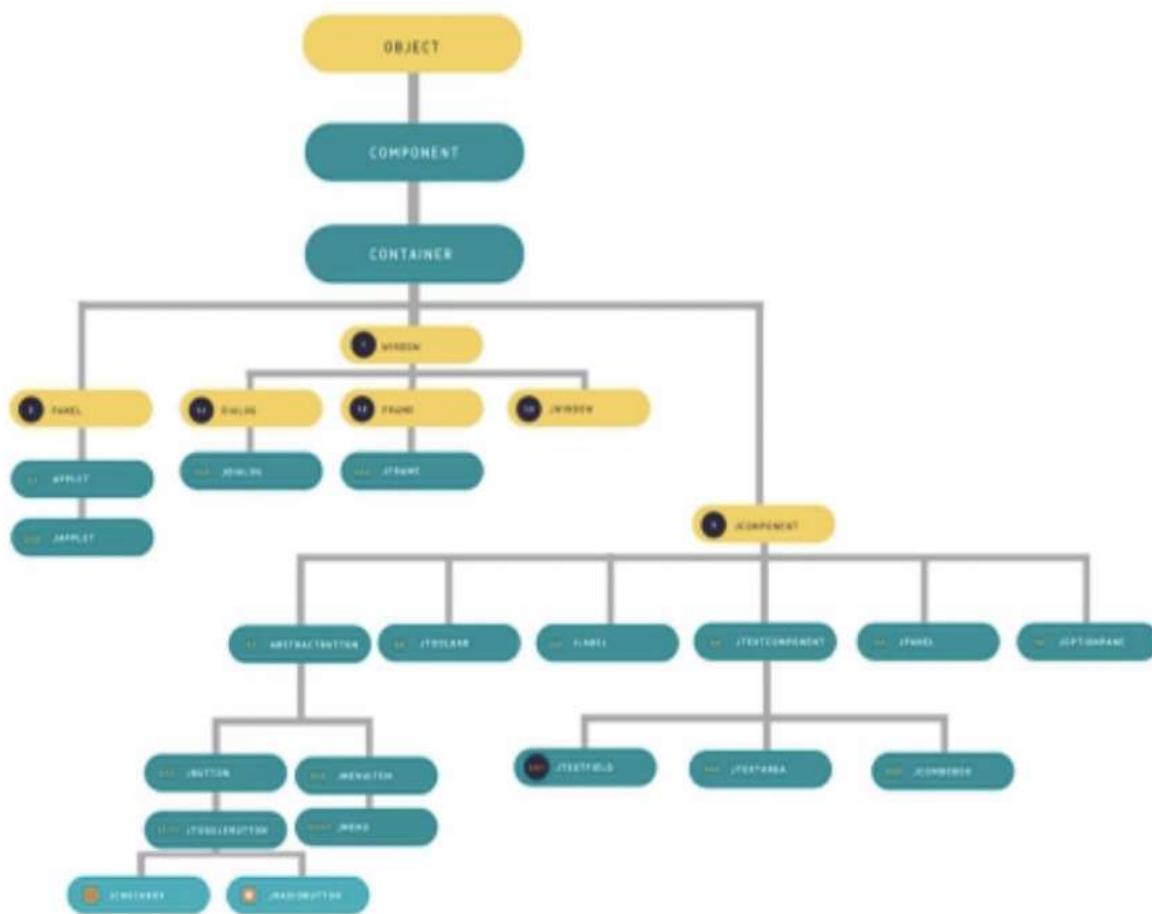
Un componente es un módulo de código ya implementado y reutilizable que puede interactuar con otros componentes software a través de las interfaces de comunicación.

#### RECUERDA

- ✓ 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 J Swing (se muestra en la figura 3.1), se podrá tomar cualquiera de ellos para desarrollar uno nuevo desde cero.



**Figura 3.1**  
Principales tipos de componentes.

### 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.

#### TOMA 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.



#### Actividad propuesta 3.1

Reflexiona sobre qué métodos clave permiten analizar el contenido de una propiedad o atributo. ¿Cuáles modifican su valor?

#### Ejemplo

Cuando hablamos de un elemento tipo ComboBox, 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	Eventos relativos a la acción del ratón sobre el componente.
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. Componentes 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

Nombre componente	Nombre evento	Descripción del evento
JTextField	ActionEvent	Detecta la pulsación de la tecla Enter tras completar un campo de texto.
[.../...]		

CUADRO 3.2 (CONT.)

JButton	ActionEvent	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.
JTextComponent	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.

CUADRO 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.

## 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.
JMenuItem	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*

Modo de pulsación	Definición
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 botones 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)
	mouseMoved	public void mouseMoved(MouseEvent e)
MouseMotionListener	mouseDragged	public void mouseDragged(MouseEvent e)

**Actividad propuesta 3.2**

Reflexiona sobre qué diferencias existen entre los eventos de tipo FocusListener y MouseMotionListener. ¿Cuál sería el más apropiado en una interfaz que sea detectar que el ratón se ha movido de arriba abajo?

### 3.3. Introspección y reflexión

El lenguaje de programación Java es estático, lo que no permite alterar ciertos aspectos de los objetos durante el tiempo de ejecución, si bien existen dos conceptos que permiten introducir ciertas funcionalidades de lenguajes dinámicos: la reflexión y la introspección.

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

La introspección es una de las características más importantes en el desarrollo de interfaces, sobre todo cuando se utilizan entornos visuales de diseño, puesto que permite a estas herramientas tomar de forma dinámica todos los métodos, propiedades o eventos asociados a un componente, que se coloca sobre el lienzo de diseño simplemente arrastrando y soltando el elemento.

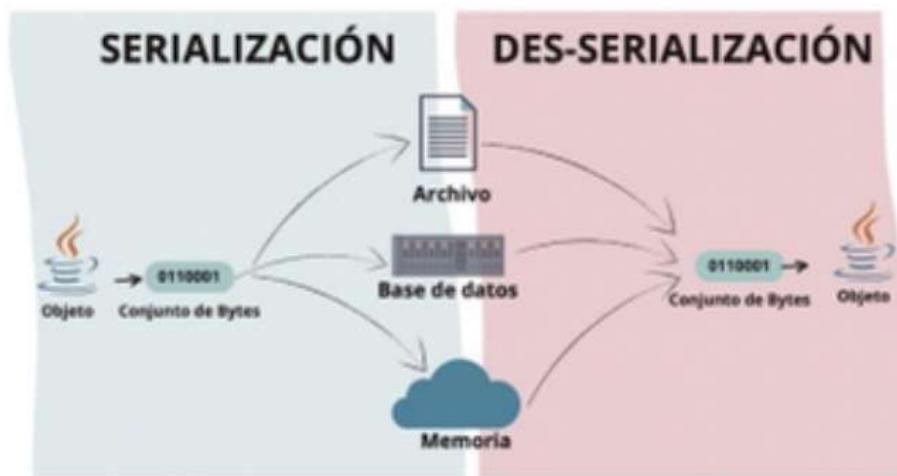
**Actividad propuesta 3.3**

Selecciona uno de los componentes disponibles en Eclipse de la librería JSwing (JTextField, JComboBox, JRadioButton...), vistos en el capítulo, y al implementarlo en la interfaz gráfica haz uso de la característica de introspección de manera que coloques directamente el elemento sobre la interfaz.

### 3.4. Persistencia del componente

El desarrollo de componentes requiere de la persistencia. Esta característica permite que el estado de una determinada clase no varíe, lo cual es posible a través de la serialización, razón por la que cuando se crea un nuevo componente será necesario implementar alguna de las interfaces siguientes en función del tipo de serialización escogida:

- a) Serialización automática. Utiliza la interfaz: java.io.Serializable.
- b) Serialización programada. Utiliza la interfaz: java.io.Externalizable.



**Figura 3.2**  
Proceso serialización.

### 3.5. Herramientas para el desarrollo de componentes visuales

A la hora de desarrollar elementos visuales para una interfaz gráfica es necesario disponer de herramientas que permitan la edición de imágenes. En función del tipo de acciones que sea necesario realizar sobre el componente será recomendable utilizar herramientas profesionales u otras más sencillas. A continuación, se analizarán algunas de las disponibles en la actualidad:

- ✓ Gimp es un programa de distribución gratuita para tareas tales como retoque fotográfico, composición de imágenes y creación de imágenes. Se puede utilizar como un programa de edición muy simple para realizar un retoque fotográfico de calidad experta, una conversión de formato de imagen...
- ✓ Microsoft Paint es muy recomendable para realizar una edición sencilla y rápida, siempre que la calidad no sea lo más importante. Para los casos en los que se necesite un desarrollo más profesional será recomendable utilizar herramientas como Photoshop. Una de las novedades más potentes que se incluyen en las últimas versiones de Paint son las funcionalidades relacionadas con el diseño 3D.
- ✓ Adobe Photoshop es uno de los programas para la edición de imágenes más popular y reconocido hoy en día. Una de las funcionalidades más importantes que incorpora esta aplicación es el uso de capas, mediante las cuales es posible aplicar a la imagen multitud de efectos y tratamientos. Algunas de sus características más importantes son:
  - Muy adecuado para el entorno profesional.
  - Permite elaborar diseños desde cero.
  - Su sistema de capas permite crear imágenes muy atractivas.
  - Es compatible con gran cantidad de tipos de formatos de imágenes.
  - Sus sistemas de filtros, efectos, eliminación del ruido, retoque de la imagen...



**Figura 3.3**  
Logotipos de herramientas de edición de imágenes (GIMP, Paint, Photoshop).

### 3.6. Empaquetado de componentes

Para lograr una distribución exitosa de una aplicación es necesario realizar paquetes donde se recoja todo lo necesario para la ejecución de una aplicación. La base de la distribución de aplicaciones se fundamenta en la creación de paquetes que incluyen todos los componentes de una aplicación software. Estos paquetes contienen todo lo necesario para desplegar una aplicación y están compuestos de ficheros ejecutables, elementos multimedia, así como librerías y bibliotecas.



**Figura 3.4**  
Diagrama empaquetado (ejemplo).



#### PARA SABER MÁS

La distribución de aplicaciones se basa en la creación de paquetes en los que se incluyen todos los componentes de diseño de una aplicación software.

El proceso de instalación de un proyecto requiere, por tanto, de un paquete software que contenga toda la información necesaria para la ejecución de una aplicación.

#### Actividad propuesta 3.4



Recopila en un solo fichero .jar todos los recursos (imágenes, sonidos...) necesarios para distribuir la interfaz utilizada en la actividad 3.4

## Resumen

- Los elementos visuales que permiten la comunicación entre el usuario y la aplicación son los conocidos como componentes visuales.
- 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. Estos elementos visuales permiten la comunicación entre el usuario y la aplicación.
- La programación basada en componentes (POC) es la metodología de programación basada en el uso de elementos reutilizables para conseguir una reutilización eficiente del software es necesario que este esté definido de la manera más generalizada posible, para poder implementar múltiples versiones modificadas.
- Las propiedades de un componente definen los datos públicos que forman la apariencia y comportamiento del objeto. Las propiedades pueden modificar su valor a través de los métodos que definen el comportamiento de un componente.
- Los métodos clave que permiten analizar el contenido de una propiedad son los de tipo get, mientras que para modificar su valor se utilizan los métodos set.
- En función del origen del evento existen dos tipos:
  - *Internos*: producidos por el propio sistema.
  - *Externos*: los producidos por el usuario.
- Los objetos que definen todos los eventos que se verán en este capítulo se basan en las siguientes clases, o en varias de estas, puesto que existe relación de herencia entre ellas.
- Con la serialización se consigue que el estado de una determinada clase no varíe, de esta manera se logra transformar los componentes en unos y ceros para almacenarlo en un fichero y así poder enviarlo, comprimirlo o poder reutilizarse en varias aplicaciones.
- Al finalizar la interfaz gráfica llega el momento de su distribución y para ello es necesario realizar paquetes donde se recoja todo lo necesario (imágenes, sonidos, librerías...) para la ejecución de una aplicación. Con la creación de paquetes que contienen todos los componentes de una aplicación software se obtiene todo lo necesario para desplegar una aplicación.

## Ejercicios propuestos



1. Además de los componentes ya predefinidos en Eclipse, es posible crear otros componentes gráficos. ¿A qué librería pertenece la clase Graphics de Java que permite realizar esta acción?

2. Para desarrollar un nuevo componente habrá que crear solo las propiedades nuevas e implementar las que ya posee el componente original.
  - a) ¿Qué tipo de propiedades crees que sería necesario añadir a un componente original como JCckBox?
  - b) ¿Y qué combinación podría realizarse utilizando un JComBox y un JTextArea?
  - c) Además del componente JCalendar, ¿qué otros componentes conoces que puedan importarse de librerías externas?
3. Importa un componente gráfico en el entorno de desarrollo Eclipse para utilizar un calendario en una interfaz gráfica. Se recomienda utilizar la librería JCalendar para poder implementar el componente JYearChooser.

## ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿Qué ventaja supone la reutilización de componentes?:
  - a) Reducción de los costes del proyecto.
  - b) Simplificación de pruebas de software.
  - c) Mejora de la calidad del software.
  - d) Todas las respuestas anteriores son ciertas.
2. ¿Qué características deben tener en cuenta a la hora de crear un componente?:
  - a) La implementación puede estar realizada con cualquier lenguaje de programación, pero ha de estar completa.
  - b) Constituye un módulo no reutilizable, sin compilar.
  - c) Su distribución se realiza en varios paquetes ejecutables.
  - d) Debe ser un elemento que no aporte funcionalidad a la interfaz.
3. ¿Cuál de las siguientes afirmaciones referida a las propiedades de los componentes es falsa?:
  - a) Las propiedades indexadas son aquellas que representan un conjunto de valores en forma de array.
  - b) Las propiedades simples son aquellas que representan solo un valor.
  - c) Los métodos clave que permiten analizar el contenido de una propiedad o atributo son los de tipo set, mientras que para modificar su valor se utilizan los métodos get.
  - d) 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.

4. Para el desarrollo de interfaces a través del lenguaje Java, existen componentes que permiten su reutilización llamados:
- a) JavaBear.
  - b) JavaTwin.
  - c) JavaBeans.
  - d) JavaPop.
5. ¿Cuál de las siguientes no es una característica común de los componentes desarrollados como JavaBean?:
- a) Parametrización.
  - b) Persistencia.
  - c) Customización.
  - d) Eventos.
6. Para definir el nuevo comportamiento de un componente e implementar nuevos métodos se usará la palabra reservada:
- a) @Superextend.
  - b) @Superride.
  - c) @Override.
  - d) @Overextend.
7. ¿Qué método devuelve un valor utilizando return?:
- a) set.
  - b) get.
  - c) Ambos.
  - d) Ninguno.
8. Al implementar un nuevo componente:
- a) Habrá que crear solo las propiedades nuevas, pero no habrá que implementar las que ya posee el componente base.
  - b) Solo podrá contener las propiedades del componente base del que hereda.
  - c) No tendrá ninguna propiedad del componente base del que hereda, siendo todas distintas.
  - d) Ninguna de las respuestas es correcta.
9. ¿Cuál de las siguientes herramientas tiene la opción de aplicar capas permitiendo aplicar a las imágenes multitud de efectos y tratamientos?:
- a) Gimp.
  - b) Paint.
  - c) Pinta.
  - d) Photoshop.
10. La clase Graphics de Java permite dibujar como si de un lienzo se tratase sobre una interfaz. Esta clase pertenece a la librería:
- a) JButton.
  - b) SWING.
  - c) AWT.
  - d) JCanvas.

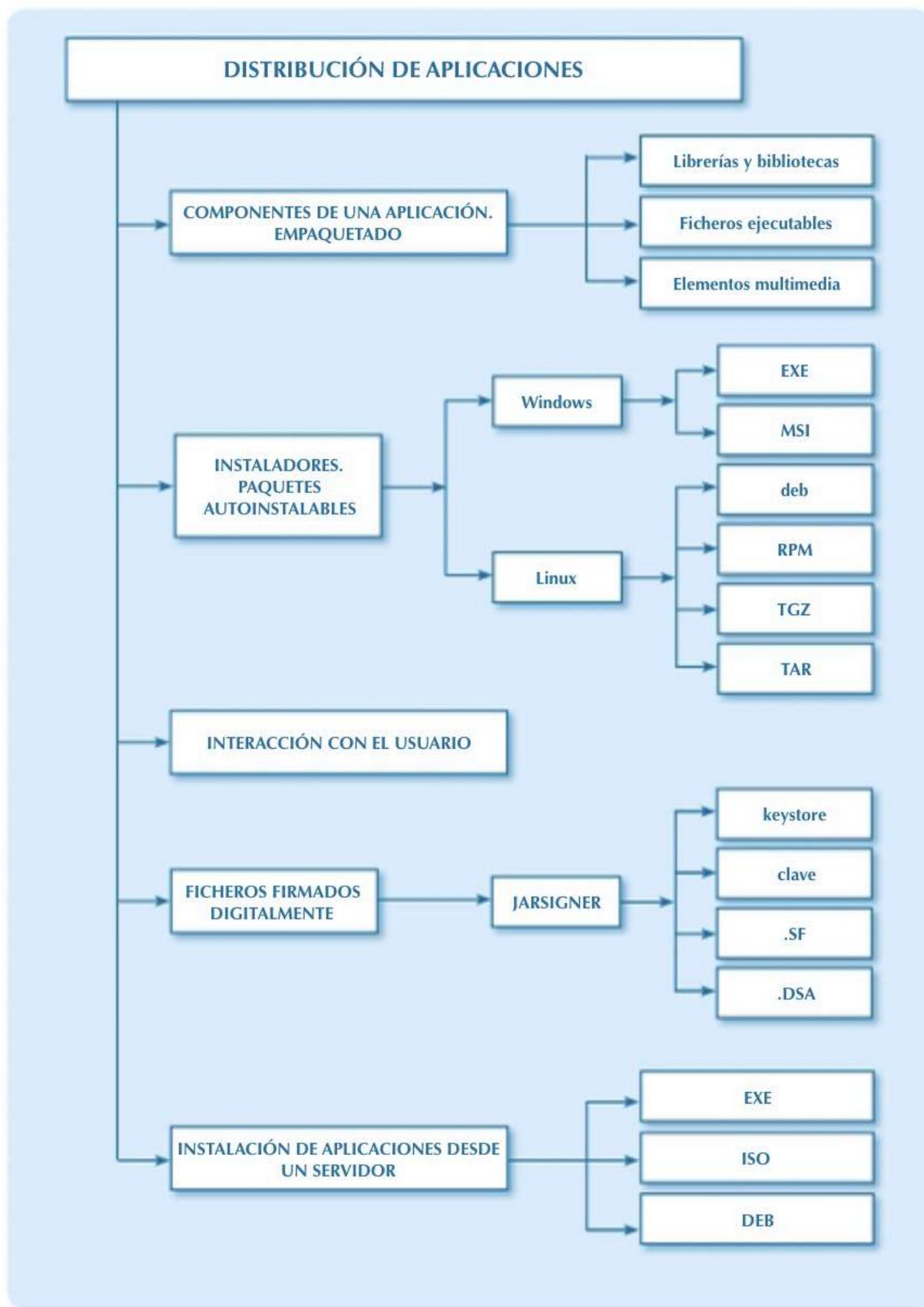
**SOLUCIONES:**1. **a b c d**2. **a b c d**3. **a b c d**4. **a b c d**5. **a b c d**6. **a b c d**7. **a b c d**8. **a b c d**9. **a b c d**10. **a b c d**

# Distribución de aplicaciones

## Objetivos

- ✓ Empaquetar los componentes que requiere la aplicación.
- ✓ Personalizar el asistente de instalación.
- ✓ Empaquetar la aplicación para ser instalada de forma típica, completa o personalizada.
- ✓ Generar paquetes de instalación.
- ✓ Preparar el paquete de desinstalación.
- ✓ Preparar la aplicación para ser descargada desde un servidor web y ejecutada.

## Mapa conceptual



## Glosario

**Clave privada.** Identificación que es conocida solo por el usuario dueño de la firma y se utiliza para firmar de forma inequívoca un fichero.

**Clave pública.** Identificación que sirve para corroborar en el destino que esta firma es auténtica.

**DEB.** Paquete usado en aquellas distribuciones que están basadas en Debian, como Ubuntu o Kubuntu.

**Empaquetado.** Agrupación de todos los componentes necesarios para el despliegue de una aplicación.

**EXE.** Archivo binario ejecutable de código reubicable.

**Instalador.** Software que lleva a cabo cada uno de los pasos necesarios del proceso de instalación. Permite la instalación de cualquier software de forma automática.

**Librería.** Conjunto de código reutilizado que permite un mayor número de funcionalidades además de disminuir la cantidad de tiempo invertido a la hora de desarrollar una aplicación.

**MSI.** Es un tipo de instalador que permite llevar a cabo una instalación de forma predefinida.

**TAR.** Paquete específico de UNIX. No tiene compresión.

**TGZ.** Paquete específico de UNIX. Se trata de paquetes TAR con compresión a través de GUNZIP.

### 4.1. Componentes de una aplicación y empaquetado

Tras implementar una aplicación llega el momento de su distribución, para ello será necesario realizar un proceso de instalación que requiere de un paso previo: el empaquetado. Para realizar un correcto empaquetado se deben agrupar todos los componentes necesarios para el despliegue de una aplicación. A continuación, se mostrarán cuáles son los principales componentes de los que debe estar compuesta una aplicación.

Un paquete contiene no solo el código que implementa una aplicación, sino que constará de todo lo necesario para desplegar una aplicación y realizar una correcta distribución. Los componentes principales son:

- Las librerías y bibliotecas.
- Los ficheros ejecutables.
- Los elementos multimedia.

El uso de las librerías permite reutilizar código y proporcionan un mayor número de funcionalidades que permiten disminuir la cantidad de tiempo invertido a la hora de desarrollar

una aplicación. Por otro lado, los elementos multimedia permiten que la interfaz sea más dinámica y por tanto la experiencia del usuario sea de mayor calidad.

#### RECUERDA

- ✓ Los ficheros ejecutables son los archivos que contienen la agrupación final del proyecto, es decir, el código, las imágenes y su posterior compresión, para que ocupen menos espacio en memoria.

## 4.2. Instaladores y paquetes autoinstalables

Para desplegar una aplicación software es necesario disponer de instaladores que lleven a cabo cada uno de los pasos necesarios del proceso de instalación. Este tipo de programas realizan una serie de operaciones sobre los archivos contenidos en el paquete de distribución que permite la instalación de cualquier software de forma automática.

En la actualidad existen multitud de instaladores, pero entre los más conocidos cabe destacar: MSI Studio, InstallBuilder y Windows Installer. Cada sistema operativo tiene diferentes características y por ello es necesario que todos los instaladores sigan una secuencia de pasos, que consiste en:

1. Comprobar las especificaciones de software y hardware del equipo.
2. Comprobar la autenticidad del software.
3. Construir los directorios necesarios para el despliegue de la aplicación.
4. Extraer los ficheros del paquete de distribución.
5. Compilar las librerías necesarias.
6. Definir las variables de entorno.



**Figura 4.1**  
Diagrama de pasos en instaladores.

### 4.2.1. Windows

Dependiendo del sistema operativo en el que nos encontramos, la forma de empaquetar las aplicaciones puede cambiar. Normalmente en el caso de Windows los más utilizados son: EXE o MSI.

- ✓ **EXE.** Este tipo de instaladores es uno de los más comunes entre los usuarios y se trata de un archivo binario ejecutable. Una de las características más destacables de este tipo de instalador es que permite seleccionar las rutas de instalación y marcar que componentes de los incluidos en el paquete se desean instalar.

- ✓ **MSI.** A diferencia de los anteriores, los paquetes MSI permiten llevar a cabo una instalación de forma predefinida. Utilizando este tipo de instaladores al usuario final se le mostrará la opción de siguiente y paso a paso será capaz de instalar la aplicación de manera preconfigurada. El desarrollador podrá seleccionar algunos aspectos de la interfaz de instalación que se le mostrará al usuario, como el texto o el ícono. En el caso de los EXE serán completamente personalizables.



#### Actividad propuesta 4.1

Busca en Internet al menos tres ejemplos de paquetes EXE y MSI. Compara las semejanzas y diferencias en el proceso de instalación de cada uno de ellos.

### 4.2.2. Linux

El sistema operativo libre por excelencia permite empaquetar y distribuir aplicaciones con diferentes formatos. En Linux se utilizan algunos tipos de paquetes que requieren de operaciones específicas a través de línea de comandos para su creación.

Entre los más destacables se encuentran:

- *deb*: usado en aquellas distribuciones que están basadas en Debian, como Ubuntu o Kubuntu. Una de las ventajas de este tipo de paquetes, a diferencia de TAR o TGZ, es que los de tipo deb pueden ser instalados directamente, mientras que los otros han de ser extraídos en primer lugar.
- *RPM*: *Redhat Package Manager*.
- *TGZ*: específico de UNIX. Se trata de paquetes TAR con compresión a través de GUNZIP.
- *TAR*: paquetes sin compresión.



#### Actividad propuesta 4.2

Realiza un empaquetado con formato DEB. Para ello debes seguir estas instrucciones:

1. Instala la distribución checkinstall para poder realizar este tipo de distribución. Utiliza la línea de comandos:

```
sudo apt-get install checkinstall
```

2. Desde la carpeta en la que se encuentran todos los ficheros del proyecto ejecuta la instrucción:

```
./configure  
make  
checkinstall
```

3. Por último, se abrirá un asistente para definir ciertos parámetros para la creación de paquetes. Se pulsará la tecla Enter y se iniciará la compilación del paquete que estará listo para su distribución.



**Figura 4.2**  
Paquetes en Linux.

### 4.3. Interacción con el usuario

A la hora de diseñar asistentes de instalación es necesario conocer el conjunto de pautas que debe seguir el desarrollador. Lo más importante es comenzar con un análisis del tipo de interacción que se va a dar entre la aplicación y el usuario antes de comenzar con su diseño. Después, se tendrá en cuenta el tipo de menús y diálogos que contendrá el asistente de instalación para la configuración de la misma siguiendo el siguiente criterio:

1. Al inicio, si la aplicación se ha desarrollado para varios idiomas, se muestra al usuario un menú para llevar a cabo la elección del idioma deseado.
2. En la actualidad es cada vez más común que para continuar con el proceso de instalación el asistente muestre la licencia de uso de la aplicación que el usuario ha de aceptar.
3. Existen aplicaciones que permiten al usuario seleccionar todas o solo algunas de las herramientas contenidas en el paquete. Se modela, por tanto, un menú que permite la selección de las mismas.
4. Después, se selecciona la ruta en la que se van a situar los archivos de la aplicación. Habitualmente, en función del sistema operativo, se utiliza una ruta por defecto, pero el usuario debe poder escoger una nueva.
5. Durante todo el proceso de instalación se muestra algún tipo de indicador del porcentaje completado sobre el total.
6. Por último, cuando el proceso concluye se le ha de notificar al usuario. En función del tipo de herramienta, puede ser necesario reiniciar el sistema operativo, en este caso se ha de preguntar al usuario si desea hacerlo en ese momento o más tarde. Del mismo modo, también suele ser habitual habilitar una opción para iniciar la ejecución de la aplicación tras finalizar el proceso de instalación.

#### Actividad propuesta 4.3



Enumera y describe cada una de las pantallas que sería necesario implementar en un asistente de instalación para una aplicación que presenta una correcta interacción con el usuario.

### 4.4. Ficheros firmados digitalmente

Tal y como se indica desde el sitio web oficial de la RAE, una firma electrónica es un conjunto de datos electrónicos que acompañan a un documento electrónico y permite identificar al firmante de forma inequívoca y asegurar la integridad del documento firmado, entre otras.

Por ello, hoy en día es necesaria la firma en la distribución de software, dado que en muchas ocasiones la descarga de nuevas aplicaciones se realiza a través de Internet, por lo que será necesario utilizar mecanismos que garanticen la autenticidad del software. Se debe conocer que las firmas digitales están formadas por una clave pública y una privada. La primera es la clave que representa la identidad de una entidad que posee la clave privada asociada a esa clave pública. La clave privada está pensada para que nunca salga del certificado y esté bajo el control del firmante, siempre.

Existen ciertas herramientas específicas que permiten el firmado digital de ficheros y que son las más aconsejadas, por ejemplo, AutoFirma utilizado para ficheros PDF o KSI Secure que permiten la firma de cualquier tipo de archivo.

En el caso de la distribución de paquetes de software, en concreto, de aquellos desarrollados utilizando Java, sería posible realizar la firma digital sobre los ficheros JAR, lo que permite verificar la autenticidad del software descargado. De esta forma, cuando se va a instalar cualquier aplicación, si se comprueba la autenticidad de la firma, se le permitirá acceder a los datos que necesite para su funcionamiento.



**Figura 4.3**  
Ficheros firmados digitalmente.

#### PARA SABER MÁS

La Fábrica Nacional de Moneda y Timbre (FNMT) emite certificados electrónicos para la identificación de aplicaciones informáticas. Existen certificados de servidor SSL, wildcard, de firma de código y de sello de entidad.

#### 4.4.1. JarSigner

Para realizar la firma de ficheros JAR se utiliza la herramienta JarSigner. Para entender el funcionamiento de este proceso se deben tener en cuenta los siguientes conceptos:

- ✓ *keystore*: se denomina así un almacén de claves en el cual puede haber contenidas muchas firmas.
- ✓ *clave*: cada par de firmas (pública y privada) están identificadas en el keystore con una clave, conocida como alias.
- ✓ *.SF*: fichero de firma. Si no especifica el nombre, utilizará las primeras ocho letras del alias en mayúsculas.
- ✓ *.DSA*: fichero del bloque de firmas. Si no especifica el nombre, utilizará las primeras ocho letras del alias en mayúsculas.

**Actividad propuesta 4.4**

Realiza la firma de un fichero JAR utilizando la herramienta JarSigner.

**CUADRO 4.1****Instrucciones JarSigner**

Opciones JarSigner	Descripción
-keystore <nombreAlmacen>	Indica el fichero keystore que se va a utilizar en cada caso, si no se indica, utiliza el almacén por defecto. La contraseña del keystore es pedida a continuación en una nueva línea por comando.
-storepass password	En este caso permite añadir la contraseña del keystore en la misma línea de comandos en la que se añade el resto de la instrucción.
-keypass password	Permite añadir la contraseña del alias en la misma línea de comandos en la que se añade el resto de la instrucción.
-sigfile file	Permite especificar el nombre de los ficheros .DSA y .SF, de lo contrario se crea utilizando el alias.
-signedjar file	Permite especificar el nombre del fichero JAR firmado. Si no se indica, se utiliza el mismo nombre que el JAR sin firmar, quedando sobreescrito.

**4.5. Instalación de aplicaciones desde un servidor web**

Hasta hace unos años, la distribución de aplicaciones software se realizaba mediante CD-ROM o archivos comprimidos almacenados en USB. Pero hoy en día gracias a los hostings o servidores web, los paquetes software pueden quedar alojados en estos servidores a los que se podrán acceder en cualquier momento para realizar la descarga a través de un conjunto de hipervínculos.

Una de las herramientas para realizar este tipo de instalación es AptUrl que permiten la descarga e instalación de paquetes alojados en un servidor web. Tras la descarga del fichero se procede a realizar la instalación del mismo. En función del tipo de instalador, como se vio al inicio de este capítulo, la instalación se realizará de diferente manera.

- EXE: se trata de un instalador que permitirá realizar la instalación ejecutando el propio fichero.
- ISO: este tipo de archivos informáticos almacenan una imagen o copia exacta del sistema de archivos. En la actualidad, los sistemas operativos incluyen opciones que permiten montar este tipo de archivos con sencillez.
- DEB: para este tipo de paquetes utilizados en Linux será necesario el uso de un gestor de paquetes para la instalación, por ejemplo, dpkg.

**TOMA NOTA**

Para crear un fichero ejecutable (instalador) de tipo EXE es posible utilizar la herramienta multiplataforma Launch4j. Con ella es posible empaquetar aplicaciones Java distribuidas como archivos JAR en ejecutables para Windows.



### Actividad propuesta 4.5

Prepara la aplicación para que pueda ser descargada y ejecutada desde un servidor web. Recuerda que previamente debe ser empaquetado y firmado como se ha visto en los apartados anteriores.

## 4.6. Creación de un asistente de instalación (instalador)

Para realizar la instalación de una aplicación, habitualmente se diseña una interfaz de usuario, también conocida como asistente de instalación, que permitirá escoger entre varias opciones de instalación (seleccionar entre diversos componentes que pueden ser instalados, la ruta concreta en la que se coloca el sistema de archivos...).

Este tipo de programas realizan una serie de operaciones sobre los archivos contenidos en el paquete de distribución que permiten la instalación de cualquier software de forma automática.

En primer lugar, para crear un asistente de instalación es necesario generar un fichero ejecutable (instalador) EXE.

### Recurso web

www

Para crear este tipo de instaladores podemos utilizar la herramienta Launch4j, disponible en [launch4j.sourceforge.net](http://launch4j.sourceforge.net)

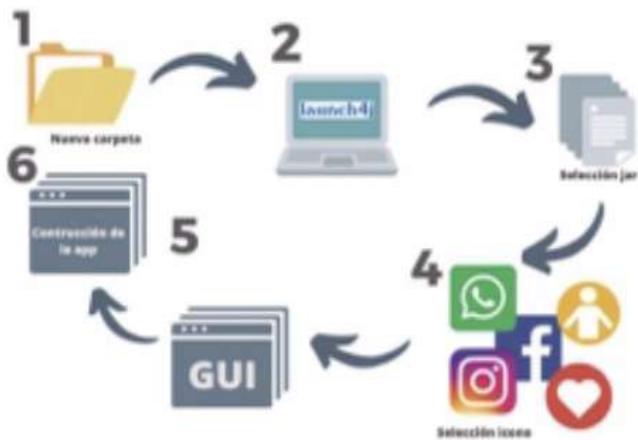
Previamente sería necesario crear el paquete JAR. Para realizar este proceso se pulsa con el botón derecho sobre el nombre del proyecto y a continuación, se escoge la opción *Export*. Seguidamente, se abrirá una *ventana de selección* y se escoge la opción *JAR File*.

El manejo de la aplicación Launch4j es sencillo. A continuación, se describen los pasos que se van a seguir:

1. Se crea una carpeta en la que se almacenará el resultado de salida del programa. La carpeta sobre la que se va a crear el fichero EXE debe contener: el paquete JAR del proyecto, librerías, imagen empleada para la creación del ícono del ejecutable y también es aconsejable incluir otra imagen, esta será la que se va a mostrar al inicio de la ejecución de la aplicación sobre la que se está construyendo el fichero EXE.
2. Ejecutamos *Launch4j* y seleccionamos la ruta en la que se ha ubicado la primera carpeta. Se indica el nombre que se le va asignar al fichero EXE.
3. En el campo *jar* seleccionamos el fichero de este tipo situado en nuestra carpeta de desarrollo del proyecto. Desde esta ventana es posible definir otras características del fichero, como la imagen que se va a utilizar como ícono del programa, a través de la opción *icon*.
4. En la pestaña *Header* seleccionamos *GUI* para que la aplicación se ejecute desde la interfaz gráfica del usuario.

5. Desde la pestaña *Splash* marcamos la casilla *Enable splash screen* y seleccionamos la imagen que se va a mostrar al usuario cuando comience la ejecución de la aplicación.
6. Finalmente, se selecciona el botón de *construcción de la aplicación*.

El proceso completo de creación del asistente de instalación, descrito previamente, se basa en el siguiente diagrama de flujo en el que se muestran los pasos que implican el proceso completo.



**Figura 4.4**  
Proceso de creación del asistente de instalación.

## Resumen

- La distribución de paquetes requiere de una correcta agrupación de todos los elementos que componen una aplicación y es lo que se conoce como proceso de empaquetado. En este desarrollo es imprescindible tener en cuenta las librerías y bibliotecas que serán necesarias, así como los ejecutables y elementos multimedia.
- Tras realizar el empaquetado es necesario que las aplicaciones puedan ser instaladas de una manera rápida y sencilla, para lo que se cuenta con los instaladores o paquetes autoinstalables. Este tipo de software realiza una serie de operaciones sobre los empaquetados de aplicaciones que permiten desplegar una aplicación en unos sencillos pasos. Dependiendo del sistema operativo que se utilice será necesario emplear los paquetes de tipo EXE o MSI en el caso de Windows o bien los de tipo .deb, .rpm, .tgz o .tar en el caso de Linux.
- En un asistente de instalación lo más importante es plantear el diseño pensando en el tipo de interacción que tendrá el usuario con la aplicación para así configurar los menús, indicadores de porcentaje de progreso o las notificaciones de finalización de instalación.
- Además de todos los pasos mencionados, es muy recomendable en el caso de la distribución de aplicaciones desarrolladas en Java, realizar la firma digital para verificar la autenticidad del software descargado. Así en el momento de la descarga se permitirá acceder a ciertos datos de manera segura. Para firmar este tipo de archivos se puede utilizar la herramienta JarSigner.
- El último paso necesario para completar el proceso de distribución de una aplicación es alojar en un servidor web los paquetes necesarios. De esta manera, se podrá acceder a los hipervínculos para después descargar e instalar el software de un modo rápido y sencillo.

## Ejercicios propuestos



1. En la instalación de aplicaciones software existen sistemas de gestión de paquetes que permiten automatizar el proceso y que variarán en función del sistema operativo o el tipo de fichero de instalación. ¿Qué tipo de componentes será necesario que contengan estos paquetes? ¿Será suficiente con el código de la aplicación?
2. Para poder crear un asistente de instalación es necesario haber generado un fichero instalador EXE. Una de las herramientas más utilizadas para la creación de ficheros de este tipo es Launch4j. Esta herramienta requiere del JAR correspondiente al proyecto sobre el que se está trabajando. Desde Eclipse escoge alguno de los proyectos ya desarrollados y crea un paquete JAR.
3. Una de las herramientas más utilizadas para la creación de asistentes de instalación es Inno Setup Compiler. Tras la creación del instalador .exe necesario para el desarrollo de un asistente de instalación, es necesario completar el proceso de construcción.

En este caso, se propone crear un asistente de instalación para alguna de las aplicaciones desarrolladas hasta ahora.

## ACTIVIDADES DE AUTOEVALUACIÓN

1. Los sistemas de gestión de paquetes permiten automatizar los procesos relativos a:
  - a) La instalación de paquetes software.
  - b) La configuración de paquetes software.
  - c) El borrado de los paquetes software.
  - d) Todas las respuestas anteriores son correctas.
2. ¿Cuál de los siguientes tipos de paquetes pueden ser instalados directamente sin necesidad de ser extraídos previamente?:
  - a) RPM.
  - b) TGZ.
  - c) deb.
  - d) TAR.
3. ¿Qué diferencia existe entre los formatos para la creación de paquetes TAR y TGZ?:
  - a) TAR es un formato utilizado para la creación de paquetes sin compresión.
  - b) TGZ es un formato utilizado para la creación de paquetes sin compresión.
  - c) TAR es un formato utilizado para la creación de paquetes con compresión a través de gunzip.
  - d) TGZ es un formato utilizado para la creación de paquetes con compresión a través de Redhat Package Manager.

4. ¿Qué paquetes podemos encontrar para el empaquetado de aplicaciones en Windows?:  
 a) MSI y AppX.  
 b) MSI y DEB.  
 c) DEB y AppX.  
 d) TGZ y TAR.
5. ¿Qué tipo de empaquetado suele realizarse de las aplicaciones en Java?:  
 a) .net  
 b) .java  
 c) .bean  
 d) .jar
6. ¿Cuál de los siguientes es un instalador muy utilizado?:  
 a) InstallBuilder.  
 b) Windows Installer.  
 c) MSI Studio.  
 d) Todas las respuestas son correctas.
7. Este instalador de Windows permite realizar la instalación de forma predefinida:  
 a) AppX.  
 b) DEB.  
 c) EXE.  
 d) MSI.
8. La distribución de aplicaciones software puede realizarse desde un servidor web. Una de las herramientas que permite la descarga e instalación de paquetes alojados en un servidor web es:  
 a) AptWeb.  
 b) AptUrl.  
 c) WebDeb.  
 d) UrlExe.
9. ¿Qué herramienta podemos utilizar para crear un tipo EXE a través de un paquete JAR?:  
 a) EnableSplash.  
 b) AptUrl.  
 c) Launch4j.  
 d) Dpkg.
10. ¿Cuál de los siguientes organismos emite certificados de firma digital?:  
 a) FNMT.  
 b) TNN.  
 c) TNT.  
 d) FETA.

**SOLUCIONES:**

1.  a  b  c  d  
 2.  a  b  c  d  
 3.  a  b  c  d  
 4.  a  b  c  d

5.  a  b  c  d  
 6.  a  b  c  d  
 7.  a  b  c  d  
 8.  a  b  c  d

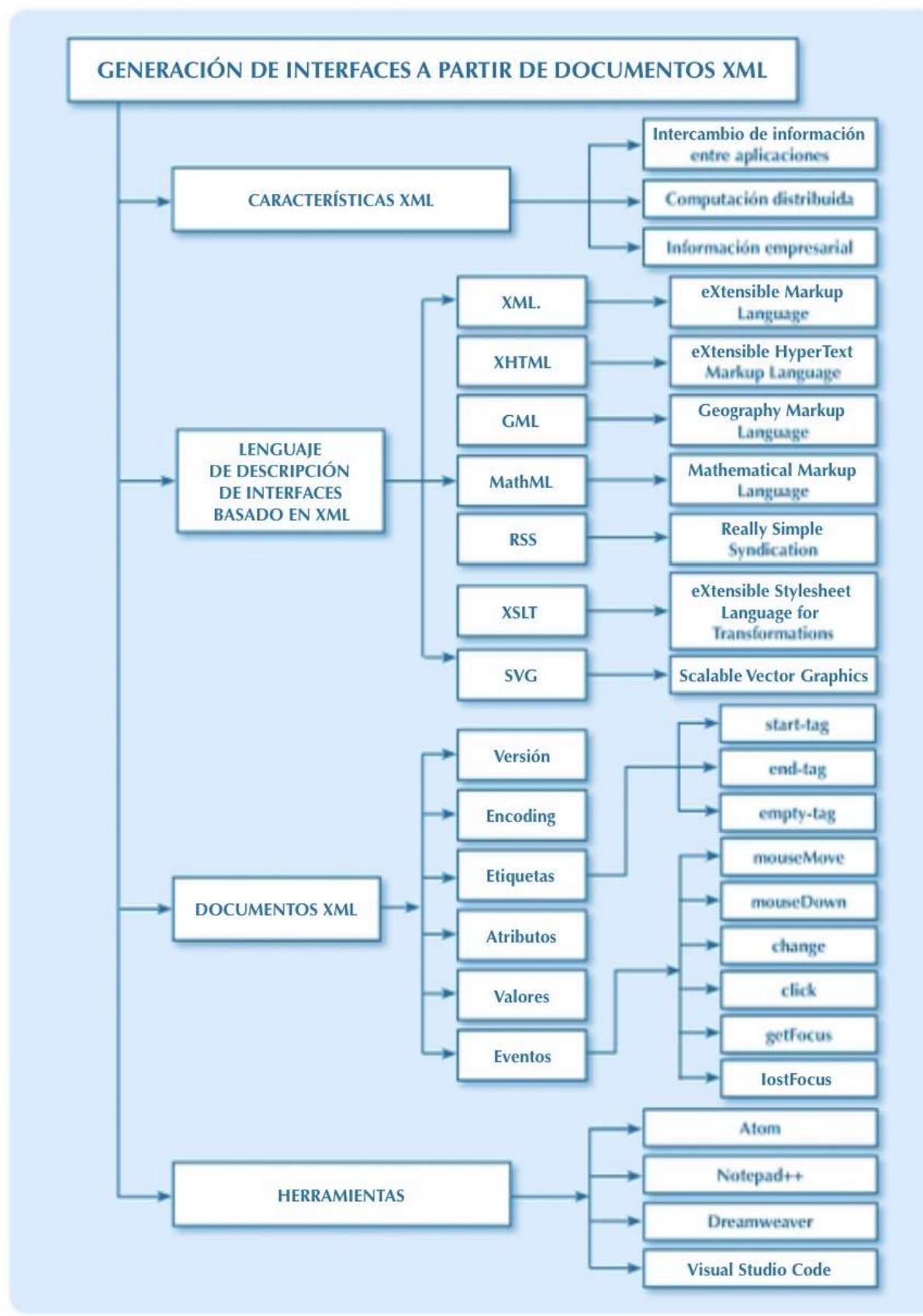
9.  a  b  c  d  
 10.  a  b  c  d

# Generación de interfaces a partir 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.

### 5.1. 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.

Este capítulo 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> — Obligatorios
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Un documento XHTML</title> — Elementos cerrado
  </head>
  <body>
    <h1>Cabecera principal del documento</h1> — Elemento en minúsculas
    <p>Nuestro primer párrafo</p>
    <h2>Cabecera secundaria</h2>
    <p>Otro párrafo con contenido distinto</p>
  </body> — Elementos anidados
</html>
```

**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 5.2**  
Implementación código GML.



#### Actividad propuesta 5.1

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 5.3**  
Implementación  
ejemplo MathML.

**Actividad propuesta 5.2**

Implementa la fórmula matemática  $a = \sqrt{c^2 - b^2}$  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</description>
        <link>https://elpais.com/ultimas-noticias/</link>
        <lastBuildDate>Mon, 06 Jan 2020 00:10:00</lastBuildDate>
        <pubDate>Mon, 06, Jan 2020 16:20:00 +0000</pubDate>
    </channel>
</rss>
```

**Figura 5.4**  
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>
```

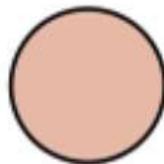
**Figura 5.5**

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 5.6**

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 5.9**

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 5.10**

Ejemplo 4: implementación de código SVG triángulo.



**Figura 5.11**  
Resultado del ejemplo 3.



**Figura 5.12**  
Resultado del ejemplo 4.

### Actividad propuesta 5.3



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:

```
<etiq1> ..... </etiq1>
```

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.

## RECUERDA

- ✓ 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_vacia />`

### 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="García-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 García-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 interesaría 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 llenar 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.



Si se desea que un texto se ponga de color rojo al situarnos encima, y de color gris al salir, existen otros eventos que podemos utilizar como *MouseEntered* y *MouseExited*; el primer evento se encargará de poner el texto de color rojo y el segundo de ponerlo de color gris.

### 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.

**CUADRO 5.1**  
Comparativa de los editores

	Atom	Notepad++	Dreamweaver	Visual Studio
Multinivel	✗	✗	✗	✗
Paneles	✗	✗	✗	✗
Control versiones	✗			✗
Libre	✗	✗	✗	✗
WYSIWYG			✗	✗

- 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.



#### TOMA 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.

- 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.

## 5.4. Generación de código para 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 basados en este lenguaje, se puede procesar para múltiples fines, como integración en una base de datos, visualización como parte de un sitio web o mensajes entre aplicaciones.
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.

**CUADRO 5.2**  
Interfaz resultado del código XML  
de la figura 5.15

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

```
<?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.

## Resumen

- El lenguaje XML permite la compatibilidad entre sistemas para compartir información de manera fácil, segura y fiable. Sus principales usos se basan en las bases de datos, editores de textos, hojas de cálculo y otras plataformas. Al ser un metalenguaje, puede ser empleado para definir otros lenguajes, entre los que destacan: XHTML, GML, MathML, RSS y SVG.
- La estructura que sigue un fichero XML se basa en etiquetas para identificar los elementos y datos que componen el documento. Estas etiquetas se deben abrir y cerrar con una estructura de este tipo: <etiq1> ..... </etiq1>
- Se debe tener en cuenta que todas las etiquetas abiertas deben haber sido cerradas en el orden adecuado. El conjunto formado por las etiquetas (apertura y finalización) y el contenido se conoce como elemento. Por otro lado, los atributos son los componentes de las etiquetas que se encuentran estructurados de manera nombre=valor. En una misma etiqueta no pueden existir dos atributos con el mismo nombre, siendo siempre todos los atributos de un elemento únicos.
- Los eventos permiten tratar las diferentes formas de interacción entre el usuario y la aplicación, tanto en las ocasiones en las que interesa tratar esta interacción como en las ocasiones que no haya una respuesta. Algunos de los eventos más comunes que se

pueden producir en una aplicación son los eventos relacionados con el movimiento del ratón, la pulsación del botón de la izquierda y la derecha del ratón o cuando se pasa sobre algún elemento como un botón de la interfaz.

- Existen herramientas que permiten crear interfaces de usuario multiplataforma que, a su vez, son editores de XML. Entre estas destacan: Atom, Notepad++, Alove Dreamweaver CCy Visual Studio Code. Con estas herramientas se logra desafiar uno de los retos más importantes ante los que se tienen que enfrentar los programadores, el intercambio de datos entre sistemas incompatibles.
- Al generar código para diferentes plataformas utilizando XML se consigue almacenar contenidos, modelar la representación de metadatos e intercambiar 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 que se esté utilizando.

## Ejercicios propuestos



1. A la hora de difundir información a usuarios que se han suscrito a una fuente de contenidos es recomendable utilizar el lenguaje de sindicación realmente simple (RSS). Busca un ejemplo en la web de utilización de este tipo de lenguaje que contenga al menos las etiquetas <title>, <link> y <description>. ¿Qué tipo de contenido se establece en cada una de ellas?
2. El siguiente código XML contiene tres errores. Localiza cada uno de ellos e indica dónde se encuentra y cómo resolverlo.
3. Una nueva empresa de suministro eléctrico nos ha solicitado el diseño de un logo que represente su marca. Esta multinacional se caracteriza por su sencillez a la hora de realizar los contratos y la rapidez a la hora de proporcionar el suministro. Desea una imagen representativa que esté formada por un círculo con un rayo en su interior. Esta imagen debe ser escalable y de gran calidad, por lo que se realizará utilizando el lenguaje SVG.

```
<?xml version="1.0" encoding="UTF-8"?>
<capitales>
<capital>
    <nombre>Pekín</nombre>
    <pais>China</pais>
</capital>
<capital>
    <nombre>Roma</nombre>
    <pais>Italia</pais>
</capital>
<capital>
    <nombre>Lisboa</nombre>
    <pais>Portugal</pais>
</capital>
<capital>
    <nombre>Lima</nombre>
    <pais>Perú</pais>
</capital>
</capitales>
```

**Figura 5.16**  
Código XML del ejercicio 2.

## ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿Qué significa XML?:

- a) X-Markup Language.
- b) eXtensible Markup Language.
- c) Example Markup Language.
- d) eXtra Modern Link.

2. ¿Cuáles es el objetivo del lenguaje RSS?:

- a) Difundir información a usuarios que se han suscrito a una fuente de contenidos actualizada frecuentemente.
- b) Representar elementos geométricos vectoriales, imágenes de mapa de bits y texto.
- c) Marcar contenido como texto, imágenes y enlaces en forma de hipervínculos para crear una estructura que pueda ser mostrada por los navegadores.
- d) Ninguna de las respuestas anteriores es correcta.

3. ¿Cuál de los siguientes códigos XHTML es erróneo?:

- a) <b><i>Texto de ejemplo</i></b>
- b) <p>Esto es un párrafo
- c) 
- d) <input type="checkbox" name="vehiculo" value="coche" checked="checked" />

4. ¿Cuál de las siguientes líneas es correcta en XML?:

- a) <i> Texto
- b) <i> Texto <i>
- c) <i> Texto </i>
- d) <l> Texto <l>

5. ¿Cuál de los siguientes sería un código XML bien formado?:

- a) <?xml version="1.0" encoding="UTF-8"?>  
<numeros>  
    <1 letra="u" letra="a" letra="d">1</>  
    <2 letra="d" letra="b" letra="e">2</>  
    <3 letra="s" letra="c" letra="f">3</>  
</numeros>
- b) <?xml version="1.0" encoding="UTF-8"?>  
<numeros>  
    <letra="u" letra="a" letra="d">1</>  
    <letra="d" letra="b" letra="e">2</>  
    <letra="s" letra="c" letra="f">3</>  
</numeros>

- c) 

```
<?xml version="1.0" encoding="UTF-8"?>
<numeros>
    <numero1 letra="a" letra2="d" letra3="o">1</1>
    <numero2 letra="b" letra2="e" letra3="s">2</2>
    <numero3 letra="c" letra2="f" letra3="i">3</3>
</numeros>
```
- d) 

```
<?xml version="1.0" encoding="UTF-8"?>
<numeros>
    <numero1 letra="a" letra2="d" letra3="o">1</numero1>
    <numero2 letra="b" letra2="e" letra3="s">2</numero2>
    <numero3 letra="c" letra2="f" letra3="i">3</numero3>
</numeros>
```

6. Un atributo en XML:

- a) Se puede encontrar en las etiquetas de cierre.
- b) La sintaxis es siempre "valor\_atributo"=nombre\_atributo.
- c) Es un componente de las etiquetas XML que consiste en un par nombre=valor.
- d) Pueden existir dos con el mismo nombre en una etiqueta.

7. Identifica cuál de las siguientes opciones es la correcta en cuanto a los valores de XML:

- a) <libro isbn=9788491713760 edicion=1 paginas=192/>
- b) <libro isbn="9788491713760" edicion="1" paginas="192"/>
- c) <libro isbn="9788491713760" edicion=1 paginas="192"/>
- d) <libro isbn=9788491713760 edicion='1' paginas=192/>

8. ¿Cuál de los siguientes no es un editor compatible con XML?:

- a) Pages.
- b) Notepad++.
- c) Dreamweaver.
- d) Visual Studio Code.

9. ¿Cuáles son las principales ventajas del editor Atom?:

- a) No es una herramienta multinivel.
- b) Su disposición de paneles es estática.
- c) No es multiplataforma.
- d) Ninguna de las respuestas es correcta.

10. ¿Por qué se genera código XML para diferentes plataformas?:

- a) Para almacenar contenidos.
- b) Para intercambiar contenidos.
- c) Para representar metadatos.
- d) Todas las respuestas son correctas.

**SOLUCIONES:**1. **a** **b** **c** **d**2. **a** **b** **c** **d**3. **a** **b** **c** **d**4. **a** **b** **c** **d**5. **a** **b** **c** **d**6. **a** **b** **c** **d**7. **a** **b** **c** **d**8. **a** **b** **c** **d**9. **a** **b** **c** **d**10. **a** **b** **c** **d**

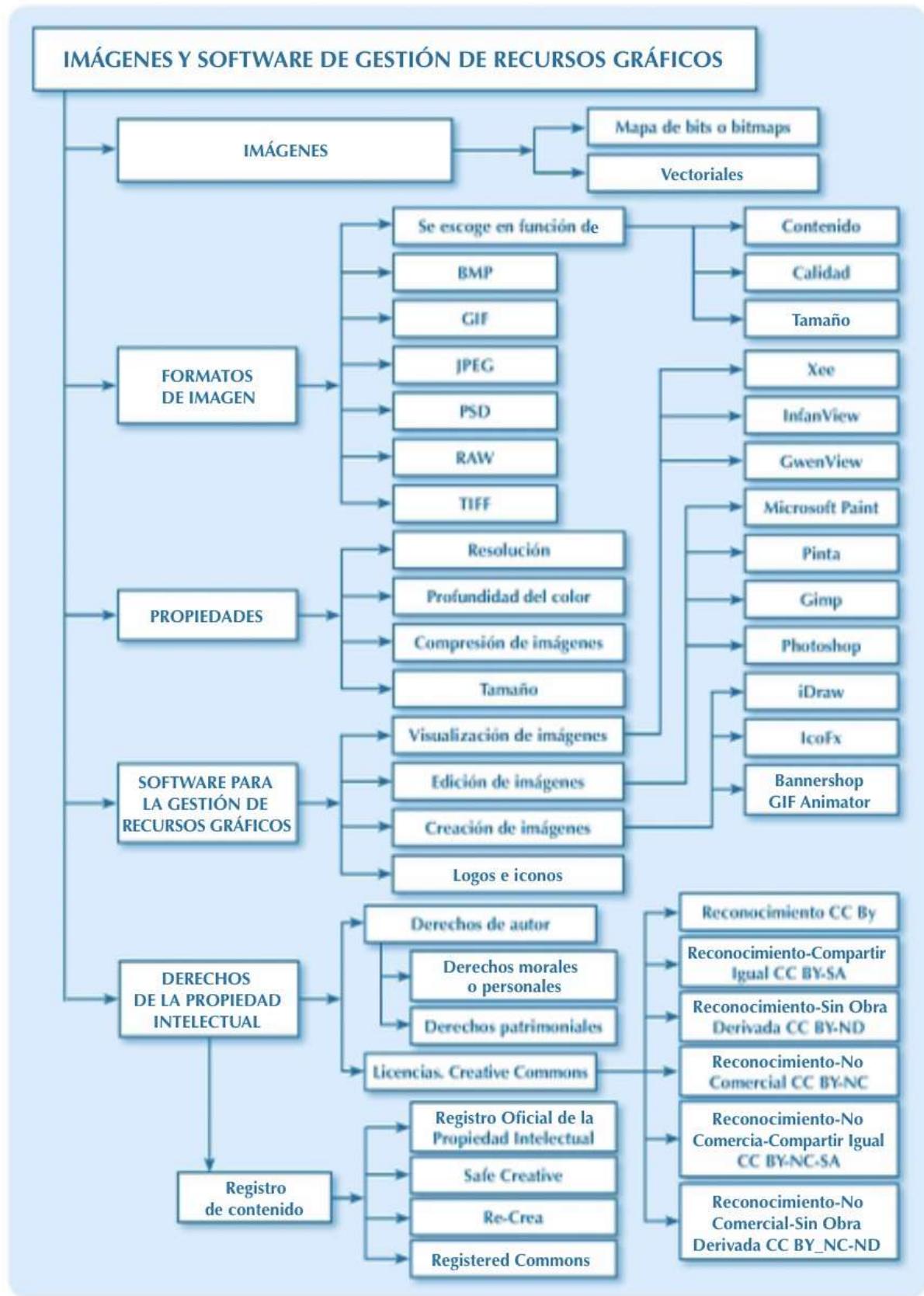
**SOLUCIONES:**1. **a b c d**2. **a b c d**3. **a b c d**4. **a b c d**5. **a b c d**6. **a b c d**7. **a b c d**8. **a b c d**9. **a b c d**10. **a b c d**

# Imágenes y software de gestión de recursos gráficos

## Objetivos

- ✓ Emplear herramientas y lenguajes específicos, siguiendo las particularidades, para desarrollar componentes multimedia.
- ✓ Identificar y proponer las acciones profesionales necesarias para dar respuesta a la accesibilidad universal y al diseño para todos.
- ✓ Identificar los formatos de imagen para utilizar en el desarrollo de una interfaz.
- ✓ Reconocer las implicaciones de las licencias y los derechos de autor en el uso de material gráfico.
- ✓ Utilizar y reconocer las tecnologías necesarias con la inclusión de imágenes en una interfaz.
- ✓ Agregar imágenes a interfaces de aplicaciones y verificar su funcionamiento.

## Mapa conceptual



## Glosario

**Compresión de la imagen.** Los sistemas de compresión utilizan un algoritmo matemático propio para reducir la cantidad de bits necesarios para describir la imagen y, por lo tanto, el peso del fichero. Se distingue entre algoritmos con pérdidas y sin pérdidas.

**Imagen digital.** Representación bidimensional de una imagen utilizando bits (unos y ceros). Dependiendo de si la resolución de la imagen es estática o dinámica, puede tratarse de un gráfico rasterizado o de un gráfico vectorial.

**Imagen vectorizada.** Representan a través de fórmulas matemáticas entidades geométricas simples y sus parámetros principales: grosor, posición inicial, final, etc. El procesador es el encargado de traducir esta información matemática a la tarjeta gráfica.

**Licencias Creative Commons.** Estas licencias proporcionan derechos de uso bajo determinadas condiciones, lo que significa que a través de unas condiciones de uso y de reconocimiento de autoría, es posible utilizar según qué contenido.

**Mapa de bits.** También llamadas imágenes rasterizadas, se trata de aquellas formadas por un conjunto de puntos, llamados píxeles, donde cada uno de estos puntos contiene un conjunto de valores que define un color uniforme.

**Profundidad de color.** Se define así al número de bits utilizados para describir el color de cada píxel de una imagen en bitmap, la cual está formada por un conjunto de píxeles, donde cada uno de ellos presenta un determinado color.

**Propiedad intelectual.** Se define así al conjunto de derechos sobre un contenido original que tienen sus autores.

**Resolución.** Grado de detalle o calidad de una imagen digital, este valor se expresa en ppp (píxeles por pulgada) o en pdi (*dots per inch*). Cuantos más píxeles contenga una imagen por pulgada lineal, mayor será su calidad.

**Software (visualización, edición y creación).** Conjunto de programas y rutinas que permiten a la computadora realizar determinadas tareas, en este caso, para la correcta visualización, edición y creación de imágenes, logos e iconos, entre otros elementos.

### 6.1. Imágenes

El uso de imágenes es muy importante en el diseño gráfico de cualquier tipo de interfaz, puesto que estas contribuyen favorablemente a la experiencia del usuario, siempre y cuando sean de calidad y se adecuen al contenido que se está trabajando. Además de cumplir ciertos requisitos de calidad y formato, se debe tener en cuenta la autoría de las imágenes, derechos de autor... Una de las características más importantes que tener en cuenta es el formato de las imágenes, puesto que de esta forma se define la calidad visual frente al peso de las ilustraciones. En este primer apartado, nos centraremos en la imagen digital en sus principales atributos.

Una imagen digital es una representación bidimensional de una imagen utilizando bits (unos y ceros). Dependiendo de si la resolución de la imagen es estática o dinámica, puede tratarse de un gráfico rasterizado o de un gráfico vectorial.

### 6.1.1. Tipos de imagen

Existen dos tipos de imágenes digitales, las vectoriales y las de mapa de bits o bitmaps, en función del tipo de aplicación en la que se vaya a emplear se escogerá un tipo u otro. La elección se basa en diferentes factores, como es el proceso de creación de las imágenes, puesto que cada una de ellas requiere de unas aplicaciones y unos requisitos determinados.

#### A) Las imágenes de mapa de bits

Estas imágenes, también llamadas de raster, son aquellas formadas por un conjunto de puntos, llamados píxeles, donde cada uno de estos puntos contiene un conjunto de valores que define un color uniforme. Por esta razón son indicadas para aquellas imágenes en las que es deseable mostrar una gama de colores muy amplia y con variaciones precisas de color y luminosidad. La calidad de estas imágenes depende de la cantidad de píxeles utilizados en su representación.

Una de las desventajas principales de las imágenes bitmap es que no permiten un cambio de escala significativo, puesto que aparece el llamado *pixelado*. Para crear o editar imágenes existen multitud de programas, algunos de software libre como Gimp, y otros para los que deberíamos adquirir su licencia de uso, como Photoshop de Adobe o Photopaint de Corel.



**Figura 6.1**

Imagen de mapa de bits o rasterizada.

#### B) Las imágenes vectoriales o de vector

Representan, a través de fórmulas matemáticas, entidades geométricas simples (puntos, segmentos, rectángulos, círculos), sus parámetros principales: grosor, posición inicial, final, etc. El procesador es el encargado de traducir esta información matemática a la tarjeta gráfica. Casi cualquier tipo de imagen puede obtenerse mediante la combinación de estas entidades geométricas más sencillas. Una de las ventajas más evidentes con respecto a las bitmaps es que pueden cambiar de escala sin perder calidad.

Para crear o editar imágenes vectorizadas existen varios programas, algunos de software libre como Sodipodi, solo disponible para plataformas Linux, y algunos más conocidos bajo licencia como Corel Draw, Adobe Ilustrator o Inkscape.



Figura 6.2  
Imagen vectorizada.



### Actividad propuesta 6.1

¿En qué situaciones de la vida cotidiana será más recomendable utilizar imágenes vectorizadas? ¿Y los mapas de bits? Para el diseño gráfico, ¿cuál crees que es la mejor opción?

#### 6.1.2. Formatos de imagen

Como se ha visto en el apartado anterior, existen dos tipos de imágenes digitales, las cuales presentan varias diferencias, entre ellas el formato en el que deben almacenarse para su posterior reproducción. Este formato aparece reflejado en la parte del nombre del fichero conocida como *extensión*. La elección de un tipo u otro, se puede basar en tres factores importantes:

1. El *contenido* de la imagen (foto, dibujo, logotipo).
2. La *calidad* que se desea obtener en función del sitio y finalidad de la publicación (publicación en web, impresión).
3. El *tamaño* que tendrá el archivo resultante.

Una de las principales decisiones a la hora de incluir gráficos en cualquier tipo de interfaz es elegir el formato correcto para cada tipo de imagen de manera que se consiga una correcta relación entre la calidad visual de la misma y su tamaño, es decir, su peso. A continuación, se definen algunos de los formatos de imagen más comunes en la actualidad:

- *BMP* Formato introducido por Microsoft y usado originariamente por el sistema operativo Windows para guardar sus imágenes.
- *GIF* (*Graphic Image File Format, formato de intercambio de gráficos*). Formato antiguo desarrollado por Compuserve con el fin de conseguir archivos de tamaño reducido. No es adecuado para imágenes fotográficas, dado que solo permite 256 colores, pero sí es indicado para otro tipo de representaciones más sencillas, tales como los logotipos. Si se almacena una imagen que tiene más de esos colores en formato GIF, se utiliza un algoritmo que aproxima los colores de la imagen a una paleta limitada por 256 colores. Por tanto, GIF es una compresión de imágenes sin pérdida solo para imágenes de 256 colores o menos. Podemos resumir sus características como:

- Número de colores: de 2 a 256.
- Formato de compresión sin pérdida basado en el algoritmo LZW.
- Carga progresiva en el navegador.
- Permite la animación simple.
- Es el formato más adecuado para aquellas imágenes sencillas, de formas simples y en las que no existe un elevado número de colores.
- *JPEG (Joint Photographic Experts Group)*. Se trata de uno de los formatos más utilizados para tratar fotografías digitales, gracias al amplio abanico de colores que admite. Es el formato utilizado en cámaras fotográficas y escáneres, por lo tanto, es el más usado en páginas web. Además, JPEG admite distintos niveles de compresión, de esta forma consigue modificar el tamaño en función del trabajo que se desee, presentando como contraprestación la disminución de la calidad. Cuanto menor sea la compresión de la imagen, mayor será la calidad, pero el tamaño de los archivos será mayor. Por el contrario, si se utiliza un nivel de compresión mayor, esta produce pérdidas y afecta a la calidad de imagen, para llevar a cabo esta reducción de tamaño, JPEG elimina la información que el ojo humano no es capaz de distinguir. Las características de este formato son:
  - Número de colores: 24 bits color u 8 bits B/N.
  - Formato de compresión con pérdida.
  - No permite la animación.
  - Por regla general, es el más indicado para aquellas imágenes que son fotografías.

**RECUERDA**

- ✓ Cada vez que se abre y manipula una foto JPEG en un ordenador, la imagen al comprimirse y descomprimirse se degrada, por lo que conviene no guardarlas en JPEG si se van a modificar. En este caso, conviene usar TIFF o BMP para editarlas y convertirlas a JPEG al final. Si no queda más remedio que editar en JPEG, hay que manipularlas con cuidado y no excesivamente.

- *PNG (Portable Network Graphics)*. Formato creado para sustituir las imágenes de formato GIF. Se trata de un sistema de compresión sin pérdida, además, permite una compresión reversible y por tanto la imagen que se recupera es exacta a la original. Las características de este formato son:
  - Color indexado hasta 256 colores y True-color hasta 48 bits por píxel.
  - Mayor compresión que el formato GIF.
  - Compresión sin pérdida.
  - No permite animación.
- *PSD*. Es el formato por defecto del editor de imágenes Adobe Photoshop y, por tanto, es un formato adecuado para editar imágenes con este programa y otros compatibles.
- *RAW*. Se trata del formato que ofrece la mayor calidad fotográfica, gracias a este tipo de formato, los píxeles no se procesan, es decir, se mantienen tal como se han tomado,

- de tal forma que pueden ser procesados posteriormente por un software específico conocido como “revelador RAW”.
- **TIFF (Tagged Image File Format).** Formato utilizado para el escaneado, la edición e impresión de imágenes fotográficas. Es compatible con casi todos los sistemas operativos y editores de imágenes.

**CUADRO 6.1**  
Comparativa de las características de cada tipo de formato de imagen

	JPEG	GIF	PNG
Colores	True-color.	256 colores.	256 colores.
Compresión	Elevada tasa de compresión. Presenta pérdidas.	Su tamaño ya es reducido.	Tasa de compresión más elevada que GIF sin introducir pérdidas.
Animación	No permite animación.	Permite animación.	No permite animación.
Usos	Aconsejable para fotografías en la web.	Diseño de logos e iconos.	Logos e imágenes.

### 6.1.3. Resolución y profundidad de color

Cuando hablamos de las imágenes estas están claramente vinculadas a dos parámetros, en primer lugar, la *resolución*, que determina el grado de detalle de la imagen, y en segundo lugar, la *profundidad de color*, la cual hace referencia al número de bits utilizado en cada píxel para describir un determinado color. Veremos a continuación ambos parámetros con más atención.

#### A) Resolución

La resolución consiste en el grado de detalle o calidad de una imagen digital. Este valor se expresa en ppp (píxeles por pulgada) o en pdi (*dots per inch*). Cuantos más píxeles contenga una imagen por pulgada lineal, mayor será su calidad. Por ejemplo, cuando hablamos de la resolución de un monitor, estamos haciendo referencia al número de píxeles por pulgada que es capaz de mostrar. Por otro lado, en un medio de impresión se habla del número de puntos por pulgada que se puede imprimir.

#### B) Profundidad de color

Una imagen en bitmap (mapa de bits) está formada por un conjunto de píxeles, donde cada uno de ellos presenta un determinado color, el archivo donde está almacenada la imagen, también contendrá la información de color de cada uno de los píxeles. El número de bits utilizados

para describir el color de cada píxel de una imagen recibe el nombre de *profundidad de color*. Cuanto mayor es la profundidad de color de una imagen, más colores tendrá la paleta disponible y, por tanto, la representación de la realidad podrá hacerse con más matices.

Si solo se dispone de 1 bit para describir el color de cada píxel, este tomará los valores 0 o 1, blanco y negro. Si disponemos de 8 bits para describir el color de cada píxel, podremos elegir entre 256 colores, puesto que como vimos en capítulos anteriores  $2^8 = 256$  colores. Esta profundidad de color es utilizada para las imágenes en modo escala de grises, desde el negro absoluto (00000000), hasta el blanco absoluto (11111111), pasando por todas las combinaciones posibles de gris.

A partir de 8 bits para profundidad, también es posible asignar colores, en concreto, 256. Entre estas posibles codificaciones de color se encuentran el negro, el blanco, grises y los colores más frecuentes. En este caso, se crea una tabla con 256 colores, cada una de las combinaciones posibles de unos y ceros de los 8 bits es un índice que permite acceder a la tabla y seleccionar un color, por esta razón, a las imágenes de 8bits se las denomina, de color indexado. Por lo tanto, cuanto mayor sea el número de bits utilizado, mayor será la profundidad de color. En el cuadro siguiente tienes el cálculo de los colores disponibles para cada profundidad:

**CUADRO 6.2**  
Relación profundidad con número de colores máximos posibles

Profundidad	Número de colores
1 bit	2
4 bit	16
8 bit	256
16 bit	65 536
32 bit	4 294 967 296

Por encima de 16 bits de profundidad, la descripción del color se divide en capas. Si la profundidad de color es de 16 bits, por ejemplo, se dedican 4 bits (128 niveles) a cada capa. Y si la profundidad es de 32 bits, cada capa utiliza 8 bits (256 niveles) para ajustar el color.

#### 6.1.4. Tamaño y compresión de imágenes

Uno de los factores más importantes a la hora de escoger las imágenes que formarán parte del diseño de una interfaz, es el tamaño de archivo de imagen, puesto que de esto dependerá la velocidad de la transferencia. Si una imagen es demasiado pesada, es recomendable utilizar formatos con compresión, como JPEG. Existen otras ocasiones en las que es deseable que el tamaño de la imagen sea elevado, lo que supondrá una mejor calidad, es el caso de la impresión fotográfica.

Por lo tanto, uno de los conceptos importantes que tener en cuenta es el tamaño del archivo, que consiste en una cifra, normalmente expresada en bits o bytes, y que cuantifica la cantidad de memoria necesaria para almacenar una imagen. Se define mediante la expresión siguiente.

$$\text{Tamaño} = R^2 * L * A * P$$

donde:

R es la resolución  
 L es la longitud de imagen  
 A es el ancho de la imagen  
 P es la profundidad del color

Finalmente, definiremos en qué consiste la *compresión de imágenes*. Tras obtener la imagen, a través del canal oportuno, esta se almacena en un fichero, compuesto por un nombre y su extensión, donde se recoge la información de la imagen, es decir, la información de cada uno de sus píxeles, necesarios para la representación de esta.

Normalmente, los archivos de tipo vector ocupan menos espacio que los de tipo mapa de bits, por lo que se hace recomendable la compresión de estos para optimizar la velocidad de procesamiento, esto hizo necesario el desarrollo de tecnologías capaces de comprimir archivos gráficos, dónde sistema de compresión utiliza un algoritmo matemático propio para reducir la cantidad de bits necesarios para describir la imagen, y marca el archivo resultante con una extensión característica: bmp, wmf, jpg, gif, png, etc.

#### TOMA NOTA



Es habitual distinguir estos sistemas en función de las pérdidas producidas en la información de la imagen durante el proceso de compresión: algoritmos con pérdidas y sin pérdidas.

## 6.2. Software para la gestión de recursos gráficos

La edición, visualización o creación de las imágenes requiere de un conjunto de herramientas básicas, desde software para la visualización de imágenes, que nos permitan operaciones sencillas como ver la imagen, ampliar ciertas zonas o ajustar algunos parámetros, tales como el brillo o la saturación. Hasta operaciones más complejas empleadas para modificar la imagen aplicando efectos, transparencias o distorsiones.

En los siguientes apartados veremos en qué consisten cada uno de esos tipos de herramientas y algunos ejemplos de aplicaciones que podemos encontrar actualmente en el mercado.

### 6.2.1. Software de visualización de imágenes

Es indispensable disponer de un software para la visualización de las imágenes. Algunas de las características más deseables en este tipo de aplicaciones son:

- ✓ Que permita aplicar y reducir el tamaño de la imagen para ser correctamente visualizada. De esta forma se pueden observar los detalles de las fotos en profundidad.

- ✓ Visualizar todas las imágenes almacenadas para poder comparar y seleccionar aquella que se acomode más a nuestras necesidades.
- ✓ Rotar y girar las imágenes, esto es, cambiar su orientación.
- ✓ Eliminar las imágenes no deseadas.
- ✓ Copiar imágenes.
- ✓ Consultar las propiedades de una imagen. Curva de color, tamaño en píxeles, etc.
- ✓ Imprimir, guardar y enviar por correo electrónico.

Existen multitud de aplicaciones para este fin, disponibles para todos los sistemas operativos Windows, Linux y Mac, como pueden ser: IrfanView, XnView y STDUViewer, para Windows, GwenView, Eye of GNOME y Feh, para Linux, FFView, Xee y Photon, para Mac.

- *Xee* (software para Mac). Xee es un visor de imágenes para los sistemas operativos OS. Se trata de un software muy ligero que no ocupa demasiada memoria. Es una de las mejores opciones. Entre sus características de funcionamiento, destacan que permite navegar fácilmente a todo el contenido de carpetas y archivos, mover y copiar imágenes rápidamente con compatibilidad de muchos formatos de archivo. Cumple su propósito de software de visualización (hacer zoom, rotar, girar) aunque no incorpora demasiadas funcionalidades extras. Es gracias a esto que se convierte en un visor ágil que no necesita demasiado tiempo de procesamiento que lo ralentice.
- *IrfanView* (software para Windows). IrfanView, desarrollado por Irfan Skiljan, fue el primer visor gráfico de Windows a nivel mundial, se trata de un software sencillo de utilizar para casi cualquier tipo de usuario, es un programa totalmente gratuito, potente y muy ligero, que permite una visualización de imágenes ágil. Además, IrfanView busca crear características únicas, nuevas e interesantes, a diferencia de algunos otros visores. Una de sus características principales es que permite automatizar tareas repetitivas o complejas. Se trata de una de las mejores opciones si se requiere modificar imágenes a menudo.
- *GwenView* (software para Linux). GwenView es un visor de imágenes y videos rápido y fácil de usar, que proporciona dos modos de funcionamiento: navegar y ver. El primero, permite la navegación por el equipo para escoger las imágenes que se van a visualizar, las cuales son mostradas como miniaturas, lo que permite escogerlas con mayor precisión. El modo ver posibilita visualizar las imágenes de una en una. La carga de imágenes se maneja a través de la biblioteca Qt, por tanto, GwenView admite todos los formatos de imagen que reconozca dicha biblioteca.



#### PARA SABER MÁS

La biblioteca Qt es un framework multiplataforma orientado a objetos, que se utiliza frecuentemente para el desarrollo de software que requiere una interfaz gráfica de usuario.

#### Actividad propuesta 6.2



Imagina que puedes diseñar tu propia aplicación para visualizar imágenes, ¿qué funcionalidad más crees necesaria?

### 6.2.2. Software de edición de imágenes

Aunque no es obligatoria la edición de imágenes, sobre todo si estas han sido tomadas con cámaras de calidad o han sido creadas como logos e iconos, es recomendable que se lleven a cabo ciertos retoques para que se mantenga un estilo común en todos los elementos que se disponen sobre una aplicación concreta.

En la actualidad existen multitud de programas que permiten retocar imágenes o crearlas como combinación de varias. A continuación, se muestran varios programas de edición que podemos encontrar en el mercado:

- ✓ *Microsoft Paint*. Una de las aplicaciones más conocidas para la edición de imágenes es la que trae instalado el sistema operativo Windows, Microsoft Paint. Se trata de un software que se utiliza para editar imágenes de una forma sencilla si bien es cierto que no permite hacer grandes cambios de diseño, para retoques sencillos, es una buena opción. Para las versiones de Windows superiores a Windows 8, existe una nueva versión de Paint, llamada Fresh Paint, que incorpora funcionalidades extra.
- ✓ *Pinta*. Es una aplicación similar a la anterior, pero en este caso se trata de software libre. Está disponible para Windows, Linux y Mac OS X. Este software ofrece las funcionalidades básicas para la edición de imágenes, así como algunas funcionalidades extra con respecto a Paint, como pueden ser los efectos.
- ✓ *Gimp*. Software multiplataforma que se encuentra disponible para Windows, Linux y Mac OS X. Se trata de una de las herramientas libres más avanzadas en la actualidad para la edición de imágenes, llegando a ser comparada con Adobe Photoshop, puesto que incorpora múltiples funcionalidades profesionales. Al incorporar más funciones, su modo de uso no es tan intuitivo como los vistos anteriormente, pero existen muchos manuales y documentación disponible para aprender a utilizarlo.
- ✓ *Photoshop*. El editor de imágenes y gráficos rasterizados por excelencia, desarrollado por Adobe System Incorporated, es utilizado para la edición de fotografías. Su potencia no solo lo ha convertido en el editor de imágenes más conocido y utilizado (sobre todo de forma profesional), sino que se utiliza para elaborar un diseño desde cero. Una de las características más destacadas son las capas en las que se subdivide la imagen, que permite aplicar diferentes efectos, textos, marcas y tratamientos a cada una de ellas.

#### Recurso web



- Pinta puede descargarse directamente desde su página web:  
<https://pinta-project.com/pintaproject/pinta/>
- Adobe System ha lanzado una versión de Photoshop, llamada Photoshop Express, una aplicación gratuita y en línea que ofrece las funcionalidades esenciales de su hermana mayor:  
<https://www.adobe.com/es/products/photoshop-express.html>

### 6.2.3. Software de creación de imágenes

En el diseño gráfico, además de la edición de imágenes, también es importante la creación de elementos que distingan nuestra imagen de marca, por ejemplo, la elaboración del logotipo de la empresa, una paleta de colores identificativa, o el diseño de iconos o botones. Algunas herramientas utilizadas para este fin son las siguientes:

- *iDraw*. Este software incorpora multitud de herramientas que permiten crear desde ilustraciones técnicas hasta imágenes, como si de obras de arte se tratara. Está disponible para Mac OS X. Al igual que Photoshop, permite la creación de capas múltiples, por otro lado, a través de una herramienta, conocida como pluma, es posible crear formas personalizadas, y permite retocar puntos de la imagen con mucha más precisión.
- *Bannershop GIF Animator* (<http://www.selteco.com/bannershop/>). Esta aplicación es utilizada para crear imágenes en formato GIF. Presenta un diseño sencillo e intuitivo que permite la creación de estos elementos de una forma ágil y dinámica.

El software para la creación de imágenes se ocupa sobre todo del diseño de los elementos visuales descritos en los apartados anteriores. Habitualmente, las imágenes y fotografías requieren de herramientas que permitan su edición y procesamiento, con el objetivo de mejorar la calidad de las mismas.

Algunas de las acciones que se pueden aplicar sobre las imágenes son la modificación del tamaño o de su posición, el borrado de elementos no deseados, el tratamiento de las propiedades, e incluso la aplicación de filtros y efectos. Un *filtro* es la transformación de una imagen digital, mediante la aplicación de operadores matemáticos. Es habitual utilizar este tipo de elementos para mejorar la calidad de una fotografía en cuanto al tratamiento de esta. Por ejemplo, permite suavizar o atenuar una imagen, realzar los bordes o eliminar la distorsión o ruido, entre otros. Mientras que un *efecto* mejora también las imágenes, en este caso añadiéndoles elementos externos, tales como sombras, reflejos, halos de color o biseles.

### 6.2.4. Software: logos e iconos

En primer lugar, hablamos de los logos o logotipos. Se trata de imágenes que caracterizan a una empresa o marca, distinguiendo de esta forma sus productos o servicios del resto. Por esta razón, realizar un buen diseño del logotipo, que resulte atractivo y fácil de identificar y asociar a la imagen de marca es clave.

Por otro lado, se encuentran los iconos gracias a los cuales es posible representar una idea, función o acción que expresada en texto ocuparía más espacio e incluso resultaría menos intuitiva para el usuario.

Por ejemplo, en muchas ocasiones puede resultar más intuitivo incluir un ícono con la imagen de una casa para regresar a la pantalla inicial que escribir “Pulsa este botón para ir a la página en la que empieza todo...”.

- *IcoFx* (<https://icofx.ro>). Editor de iconos gratuito, que permite la creación, extracción y edición de iconos. Está disponible para sistemas sobre Windows, a partir de XP, y para Mac OS X. Además de la creación de iconos desde cero, permite convertir imágenes

en iconos. Es deseable que los iconos de una misma interfaz de aplicación presenten características similares, IcoFX permite crear librerías de iconos.



### Actividad propuesta 6.3

Imagina que vas a comenzar el diseño gráfico de tu aplicación, y tienes que preparar el paquete de herramientas software necesario, en base a los tipos expuestos, ¿qué programa o programas escogerías?, ¿cuál consideras imprescindible?

El abanico de posibilidades es muy amplio, ¿conoces alguna aplicación más que consideres interesante? O ¿alguna función más para las vistas que no se haya descrito?

## 6.3. Las imágenes y la ley de propiedad intelectual

Las imágenes que se utilizan en cualquier tipo de diseño deben ser correctamente obtenidas, es decir, o bien son de elaboración propia y tenemos sus derechos de uso, o si utilizamos otras deben estar correctamente referenciadas. En este apartado nos centraremos en los puntos relativos a este tema.

### 6.3.1. Derechos de la propiedad intelectual

¿Qué es la propiedad intelectual? Es posible encontrar diversas definiciones sobre este controvertido concepto, que podemos resumir “como el conjunto de derechos sobre un contenido original que tienen sus autores”.

De forma más genérica, una de las acepciones más dominante en la actualidad en gran variedad de países es la definición recogida por la OMPI (Organización Mundial de la Propiedad Intelectual), la cual define la propiedad intelectual como:

La propiedad intelectual (P.I.) tiene que ver con las creaciones de la mente: las invenciones, las obras literarias y artísticas, los símbolos, los nombres, las imágenes y los dibujos y modelos utilizados en el comercio.

La propiedad intelectual se divide en dos categorías: la propiedad industrial, que incluye las invenciones, patentes, marcas, dibujos y modelos industriales e indicaciones geográficas de procedencia; y el derecho de autor, que abarca las obras literarias y artísticas, tales como las novelas, los poemas y las obras de teatro, las películas, las obras musicales, las obras de arte, tales como los dibujos, pinturas, fotografías y esculturas, y los diseños arquitectónicos. Los derechos relacionados con el derecho de autor son los llamados derechos conexos de los artistas intérpretes y ejecutantes sobre sus interpretaciones y ejecuciones, los derechos de los productores de fonogramas sobre sus grabaciones y los derechos de los organismos de radio-difusión sobre sus programas de radio y de televisión.



## TOMA NOTA

Es importante establecer la diferencia que existe entre la propiedad intelectual y el derecho de autor:

- La propiedad intelectual es la normativa que recoge el conjunto de derechos morales y patrimoniales que corresponden a los autores respecto de las creaciones intelectuales provenientes de su esfuerzo o trabajo, dignos de reconocimiento jurídico. Mientras que el derecho de autor es la protección que le otorga la Ley de Propiedad Intelectual al autor de la obra, por el solo hecho de su creación.
- El derecho de autor comprende derechos de naturaleza moral y patrimonial.

### 6.3.2. Derechos de autor

En segundo lugar, otro de los términos importantes a tener en cuenta son los derechos de autor, podemos definirlos como el conjunto de normas y principios que regulan los derechos de los autores, sobre cualquier tipo de obra creada por estos, es decir, desde que se crea una obra, el autor posee plenos derechos sobre la misma. Estos derechos están constituidos por dos claves: derechos morales y derechos patrimoniales.

- a) *Derechos morales o personales*: los cuales incluyen aspectos sobre el reconocimiento de la condición de autor de la obra.
- b) *Derechos patrimoniales*: estos son susceptibles de tener un valor económico y suelen estar asociados al concepto anglosajón de Copyright.

#### Actividad propuesta 6.4



¿Qué diferencias principales crees que existen entre los derechos de la propiedad intelectual y los derechos de autor? ¿Consideras necesaria la distinción que se hace entre ambas?

### 6.3.3. Licencias

Las licencias Creative Commons proporcionan ciertos derechos de uso bajo determinadas condiciones, es decir, no significa que no tengan derechos de autor, sino que a través de unas determinadas condiciones de uso, y de reconocimiento de autoría, es posible utilizar según qué contenido. En función de la elección de la licencia por parte del autor de la obra, será posible prohibir la reproduc-

ción y distribución de la totalidad o parte de esta sin la autorización expresa del autor. Es posible que el autor decida poner a disposición del público su obra, esta deberá autorizarse explícitamente para cada uso que vaya a hacerse de ella o se estará vulnerando la ley.

En base a esto podemos distinguir entre diferentes tipos de licencias Creative Commons que se clasifican en cuatro grandes licencias: atribución de la obra, no comercial, sin derivados y compartir igual. Las combinaciones posibles de licencias y su función se describen a continuación:

**CUADRO 6.3**  
**Tipos de licencias Creative Commons**

Tipo de licencia	Descripción	Imagen
Reconocimiento CC By	Esta licencia permite a otros distribuir, mezclar, ajustar y construir a partir de su obra, incluso con fines comerciales, siempre que le sea reconocida la autoría de la creación original.	
Reconocimiento- Compartirlugal CC BY-SA	Esta licencia permite a otros modificar y desarrollar sobre una obra, incluso para propósitos comerciales, siempre que te atribuyan el crédito y licencien sus nuevas obras bajo idénticos términos. Cualquier obra nueva basada en la tuya, lo será bajo la misma licencia, de modo que cualquier obra derivada permitirá también su uso comercial.	
Reconocimiento- SinObraDerivada CC BY-ND	Esta licencia permite la redistribución, comercial y no comercial, siempre y cuando la obra no se modifique y se transmita en su totalidad, reconociendo su autoría.	
Reconocimiento- NoComercial CC BY-NC	Esta licencia permite a otros ajustar y construir a partir de su obra con fines no comerciales, y aunque en sus nuevas creaciones deban reconocer su autoría y no puedan ser utilizadas de manera comercial, no tienen que estar bajo una licencia con los mismos términos.	
Reconocimiento- NoComercial- Compartirlugal CC BY-NC-SA	Esta licencia permite a otros ajustar y construir a partir de su obra con fines no comerciales, siempre y cuando le reconozcan la autoría y sus nuevas creaciones estén bajo una licencia con los mismos términos.	
Reconocimiento- NoComercial- SinObraDerivada CC BY-NC-ND	Esta licencia es la más restrictiva, solo permite que otros puedan descargar las obras y compartirlas con otras personas, siempre que se reconozca su autoría, pero no se pueden cambiar de ninguna manera ni se pueden utilizar comercialmente.	

### Recurso web

www

Para más información, se aconseja consultar el sitio web de Creative Commons: <https://creativecommons.org>

### Actividad propuesta 6.5



Ante las siguientes situaciones, reflexiona sobre cuál crees que es la licencia que más aconsejarías en cada caso.

- a) Una tabla como la mostrada en el cuadro 6.2.
- b) Una imagen fotográfica tomada por nosotros como diseñadores gráficos de la aplicación y que va a ser utilizada como logotipo identificando inequívocamente la marca.
- c) Un aplicación completa diseñada por una empresa de desarrollo de interfaces.

#### 6.3.4. Registro de contenido

El registro es un medio que garantiza la protección de los derechos de la propiedad intelectual, su inscripción no es obligatoria. El autor decide cuándo y por qué quiere registrar una obra.

Dicho registro es una constancia de la autoría sobre una obra, no impide que estas sean plagiadas o se cometan otras infracciones sobre ellas, debemos tener en cuenta que lo que es registrado es una obra y no una idea, para estas últimas se utilizan las conocidas como patentes y marcas. Existen varios sistemas de propiedad intelectual a través de los cuales es posible llevar a cabo el registro de una obra. Para el caso de los registros privados de la propiedad intelectual, que facilitan la inscripción y publicación del autor y titulares de los derechos, este proceso se garantiza mediante un sistema de huella digital y time-stamping o sello de tiempo.

- ✓ *Registro Oficial de la Propiedad Intelectual.* Se trata de un registro público, que suele existir en todos los países. En España, el Registro General de la Propiedad Intelectual es único en todo el territorio nacional y es un mecanismo administrativo para la protección de los derechos de propiedad intelectual de los autores y demás titulares sobre sus obras, actuaciones o producciones.
- ✓ *Safe Creative.* Registro privado de la propiedad intelectual. Se trata de un registro o depósito de obras de propiedad intelectual en formato digital. Como se expone desde su sitio web, Safe Creative es una empresa que lleva desde el año 2007 ofreciendo los sistemas tecnológicos para la generación y gestión de evidencias de autoría y derechos relacionados más innovadores, eficientes y avanzados. Cuenta con el aval de decenas de miles de creadores, empresas e instituciones alrededor del mundo, se ha convertido en interlocutor habitual y referencia con relación a políticas y otros aspectos relacionados con la propiedad intelectual.
- ✓ *Re-Crea.* Es un depósito de creaciones donde el usuario envía su documento en línea a un servidor seguro de la Cámara de Comercio de Barcelona y automáticamente se genera un sello de tiempo y un certificado conforme el documento ha sido depositado en una hora y una fecha concreta. Re-Crea impide la manipulación, por cualquiera de las partes, del documento depositado.
- ✓ *Registered Commons.* Depósito de creaciones. Este registro permite explotar económicamente la obra a través de la plataforma, para lo que realiza venta de licencias.

## Resumen

- El uso de imágenes es muy importante en el diseño gráfico, puesto que estas contribuyen favorablemente a la satisfacción del usuario, siempre y cuando sean de calidad y se adecuen al contenido que se está trabajando. Además de cumplir ciertos requisitos de calidad y formato, se debe tener en cuenta la autoría de las imágenes, derechos de autor, etc. Existen dos tipos de imágenes digitales principales, estas son las *vectoriales* y las de *mapa de bits o bitmaps*, en función del tipo de aplicación en la que se vaya a emplear escogeremos un tipo u otro.
- Una de las principales decisiones a la hora de incluir gráficos en el desarrollo de una interfaz para una aplicación es elegir el formato correcto para cada tipo de imagen de manera que se consiga una correcta relación entre la calidad visual de la misma y su tamaño, es decir, su peso. Algunos de los formatos de imagen más comunes en la actualidad: *BMP*, *GIF* (*Graphic Image File Format, Formato de intercambio de gráficos*), *JPEG* (*Joint Photographic Experts Group*) o *PNG* (*Portable Network Graphics*).
- La resolución consiste en el grado de detalle o calidad de una imagen digital. Este valor se expresa en *ppp* (*píxeles por pulgada*) o en *pdi* (*dots per inch*). Cuantos más píxeles contenga una imagen por pulgada lineal, mayor será su calidad.
- Una imagen en bitmap está formada por un conjunto de píxeles, en el que cada uno de ellos presenta un determinado color. El archivo donde está almacenada la imagen también contendrá la información de color de cada uno de los píxeles. El número de bits utilizados para describir el color de cada píxel de una imagen recibe el nombre de *profundidad de color*.
- La edición, visualización o creación de las imágenes requiere de un conjunto de herramientas básicas, desde software para la visualización de imágenes, que nos permitan operaciones sencillas como ver la imagen, ampliar ciertas zonas o ajustar algunos parámetros, tales como el brillo o la saturación; hasta operaciones más complejas empleadas para modificar la imagen aplicando efectos, transparencias, o distorsiones.
- Finalmente, veremos brevemente en este resumen los derechos de autor y otros aspectos relacionados. Las imágenes que se utilizan en una aplicación deben ser correctamente obtenidas, es decir, o bien son de elaboración propia y tenemos sus derechos de uso, o si utilizamos otras deben estar correctamente referenciadas. Podemos definir la propiedad intelectual “como el conjunto de derechos sobre un contenido original que tienen sus autores”.
- Otro de los términos importantes son los derechos de autor, que podemos definir como el conjunto de normas y principios que regulan los derechos de los autores sobre cualquier tipo de obra creada por estos, es decir, desde que se crea una obra, el autor posee plenos derechos sobre la misma.
- Las licencias Creative Commons proporcionan ciertos derechos de uso bajo determinadas condiciones, es decir, no significa que no tengan derechos de autor, sino que a través de unas determinadas condiciones de uso y de reconocimiento de autoría es posible utilizar según qué contenido. En función de la elección de la licencia por parte del autor de la obra, será posible prohibir la reproducción y distribución de la totalidad o parte de esta sin la autorización expresa del autor.
- El registro es un medio que garantiza la protección de los derechos de la propiedad intelectual, su inscripción no es obligatoria. El autor decide cuándo y por qué quiere registrar una obra.



## Ejercicios propuestos

1. Escoge algunas de las herramientas vistas a lo largo del capítulo, o puedes utilizar alguna otra que conozcas. Realiza el diseño y creación de un logo que identifique alguna de las aplicaciones que has ido desarrollando a lo largo del libro/curso. Este tiene que presentar la calidad suficiente como para ser utilizado. ¿Qué elementos son los que más se valoran al crear en diseño un logotipo para una aplicación?
2. Busca dos imágenes, la primera de tipo mapa de bits y la segunda vectorizada, ambas deben tener una licencia Creative Commons Reconocimiento CC By. ¿Te ha resultado muy complicado encontrarlas? ¿Has localizado algún repositorio de imágenes que presente únicamente licencias de este tipo? Te aconsejo visitar algunos como Pixabay o Unsplash.
3. Imagina que has completado el diseño de la interfaz de una aplicación completa, compuesto por logos de elaboración propia, imágenes tomadas exclusivamente para la elaboración de la aplicación y textos originales. ¿Qué pasos seguirías en cuanto a licencias, derechos de autor y propiedad intelectual?

## ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿Qué tipo de derechos posee un autor sobre sus creaciones sean estas obras literarias, musicales, teatrales, artísticas, científicas o audiovisuales?:
  - a) Derechos morales.
  - b) Derechos de autor.
  - c) Derechos sobre la propiedad intelectual.
  - d) Derechos literarios.
2. Las imágenes almacenadas en formato PNG:
  - a) Están almacenadas usando compresión con pérdida.
  - b) Es posible recuperar una imagen exacta a la original tras el proceso de compresión.
  - c) Permite animaciones.
  - d) Son las antecesoras del formato GIF.
3. A diferencia de Photoshop, Gimp es una herramienta que:
  - a) Requiere de conocimientos profesionales en edición para su uso.
  - b) Es gratuita.
  - c) Solo está disponible para Linux.
  - d) Ocupa mucho espacio en la memoria del ordenador.

4. La elección de un tipo u otro de formato de imagen se puede basar en tres factores importantes:
- a) Calidad, tamaño y resolución.
  - b) Calidad, contenido y profundidad de color.
  - c) Calidad, contenido y tamaño.
  - d) Contenido, tamaño y profundidad de color.
5. ¿Qué atributo define el grado de detalle o calidad de una imagen digital?:
- a) La resolución de una imagen.
  - b) La profundidad de color de una imagen.
  - c) El grado de nitidez de una imagen.
  - d) El tamaño de una imagen.
6. ¿Cuál de las siguientes aplicaciones está recomendada para la creación de logos o iconos?:
- a) Gimp.
  - b) iDraw.
  - c) Photoshop.
  - d) Microsoft Paint.
7. El contenido de la propiedad intelectual está formado por:
- a) Derechos de carácter personal y patrimonial.
  - b) Un conjunto de toda clase de derechos.
  - c) Los derechos recogidos en el Registro de la Propiedad Intelectual, tras realizar la inscripción en el mismo.
  - d) Ninguna de las respuestas anteriores es correcta.
8. ¿Qué es la OMPI?:
- a) La Organización Mundial de Propiedad Intelectual.
  - b) La Orden Mundial de Propiedad Intelectual.
  - c) La Organización Multimarca de Propiedad Intelectual.
  - d) La Organización Mundial del Poder Intelectual.
9. La propiedad intelectual de una obra nace:
- a) En el momento en que se inscribe esa obra literaria, en el Registro de la Propiedad Intelectual.
  - b) Una vez ha sido divulgada la obra.
  - c) Despues de 24 horas tras la inscripción de la obra en el Registro de la Propiedad Intelectual.
  - d) Por el solo hecho de su creación.
10. En cuanto a los registros de la propiedad intelectual:
- a) Solo son válidos los registros públicos.
  - b) Existe un solo registro a nivel mundial.
  - c) Re-Crea es un registro de tipo privado.
  - d) Es obligatorio registrar cualquier tipo de obra.

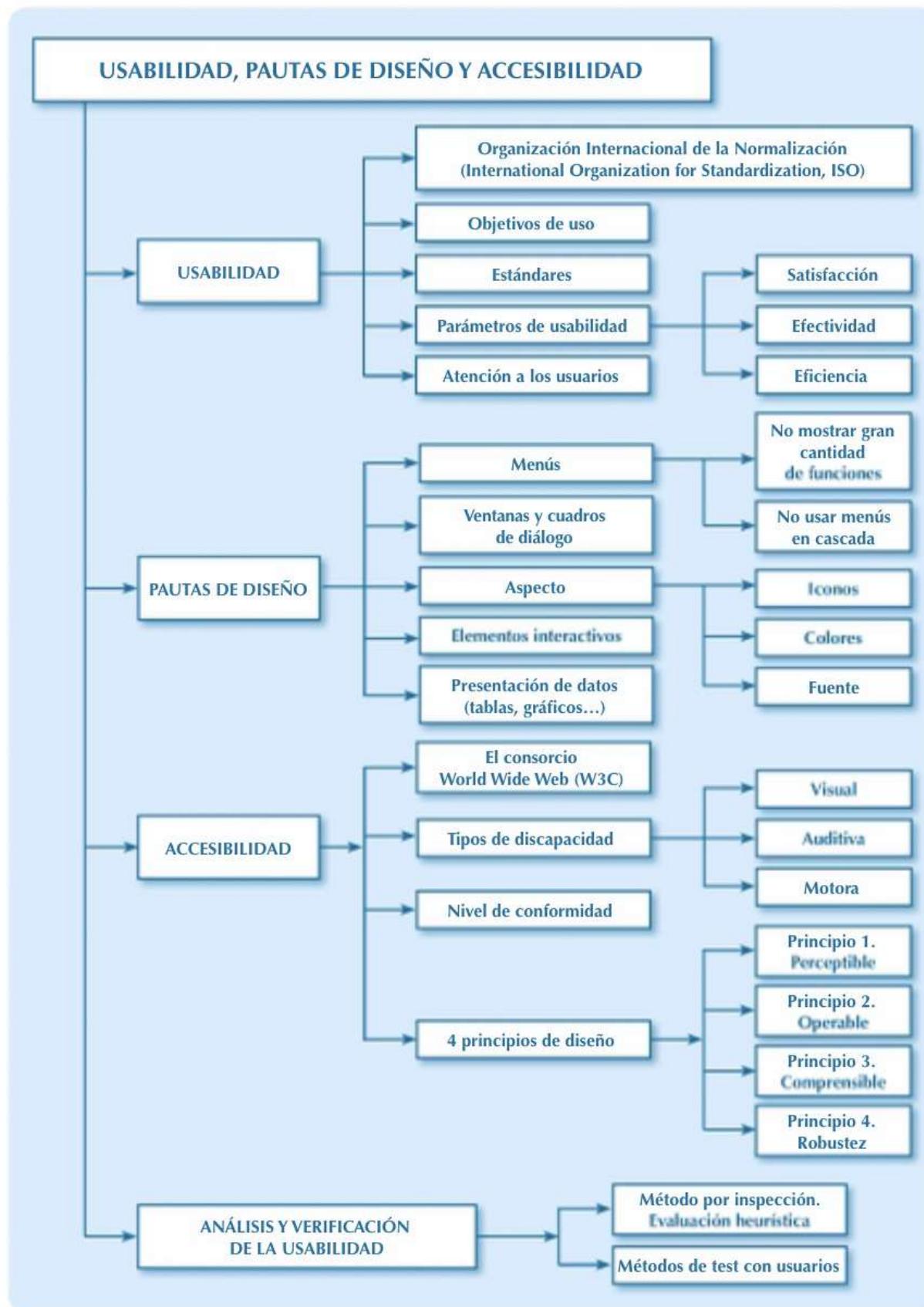
**SOLUCIONES:**1. **a b c d**2. **a b c d**3. **a b c d**4. **a b c d**5. **a b c d**6. **a b c d**7. **a b c d**8. **a b c d**9. **a b c d**10. **a b c d**

# Usabilidad, pautas de diseño 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



## 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.** Se define así la dificultad que presentan algunas personas para participar 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.

### 7.1. 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 7.1



¿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?

#### 7.1.1. 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 determinando 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.

## TOMA NOTA



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 7.1**  
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 7.2



¿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.



**Figura 7.2**  
Medidas de usabilidad.

- ✓ *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.

### 7.1.2. 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 lingüística 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 considerar 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 7.1**  
**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.

## 7.2. 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:

### 7.2.1. 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.

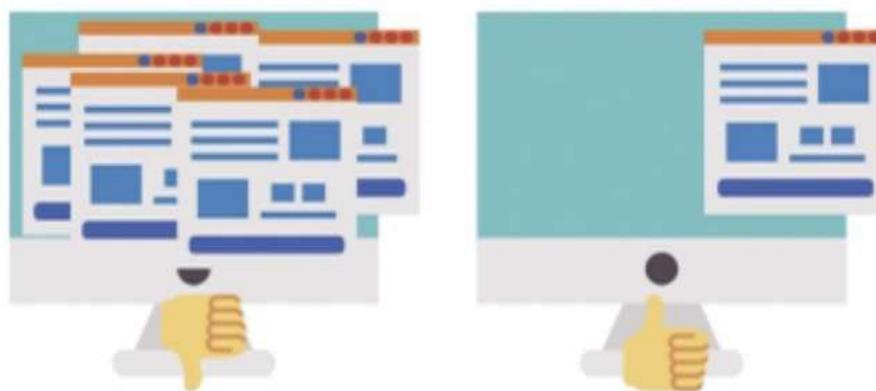
Otra de las pautas de diseño más importantes es definir una zona concreta en la que se mostrarán las diferentes opciones y que no variará a lo largo del diseño, ya que, de lo contrario, el usuario deberá buscar en cada nueva ventana donde ha sido colocado el menú disminuyendo la satisfacción de uso sobre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

### 7.2.2. 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 7.3**  
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 7.4**  
Cuadro de diálogo claro y diálogos sin posibilidad de solución.

### 7.2.3. Pautas de diseño relativas al aspecto

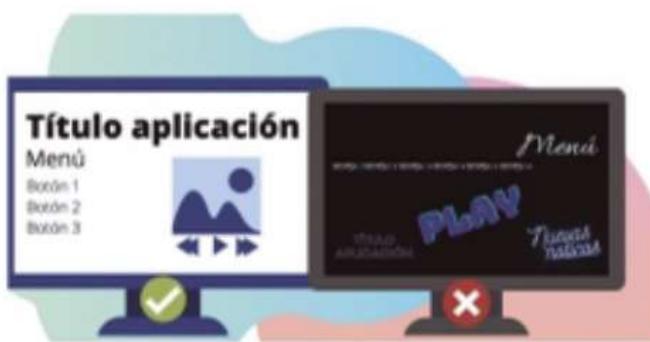
El diseño relativo al aspecto de la interfaz de usuario pone el foco en los elementos esenciales: color, fuente y distribución de los elementos. El diseño de los elementos visuales debe facilitar y mejorar la usabilidad de la aplicación, una mala selección puede llevar al fracaso a la mejor de las aplicaciones.

- 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 7.5**  
Ejemplos de psicología 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 el 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.

#### 7.2.4. 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:

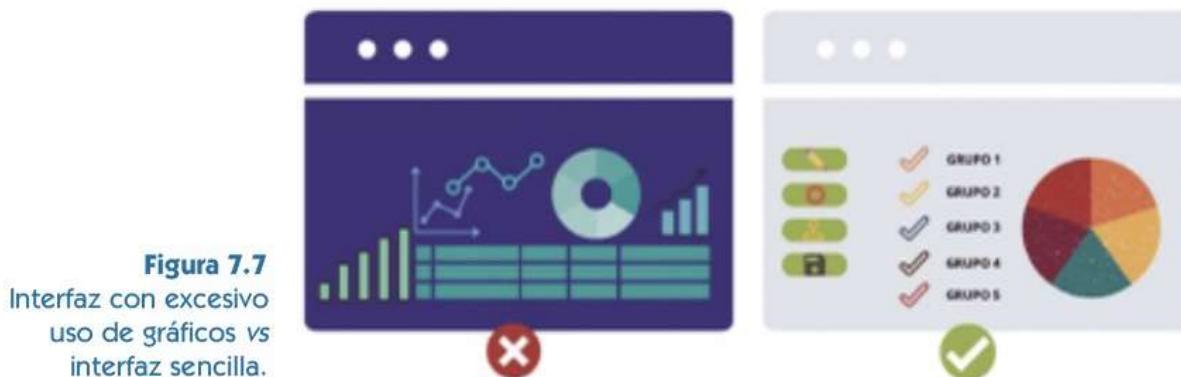
1. Los títulos deben ser *intuitivos*.
2. Las acciones codificadas en cada opción deben quedar lo suficientemente *comprendibles para el usuario*.
3. Las opciones deben ser *fácilmente distinguibles* unas de otras y, por tanto, relativamente rápidas de escoger y seleccionar.

### 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 7.7**  
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.

### 7.3. 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.

#### 7.3.1. 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 cuatro 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.

## TOMA NOTA



- ✓ *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 ampliadores 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.



### Actividad propuesta 7.3

¿Qué harías para adaptar la presencia de captchas a usuarios con dificultades visuales? ¿Crees que existe alguna recomendación internacional en la que se recojan directrices para solucionar este tipo de barreras?

### 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 interactuar 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.



#### PARA SABER MÁS

Si es necesario pulsar diversas teclas al mismo tiempo y no es posible utilizar las dos manos, los sistemas operativos ya incorporan software que permite crear el denominado "StickyKeys". Este sistema consiste en la creación de patrones en los que la repetición de la pulsación de una determinada tecla equivale a combinaciones de teclas.

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. A 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.

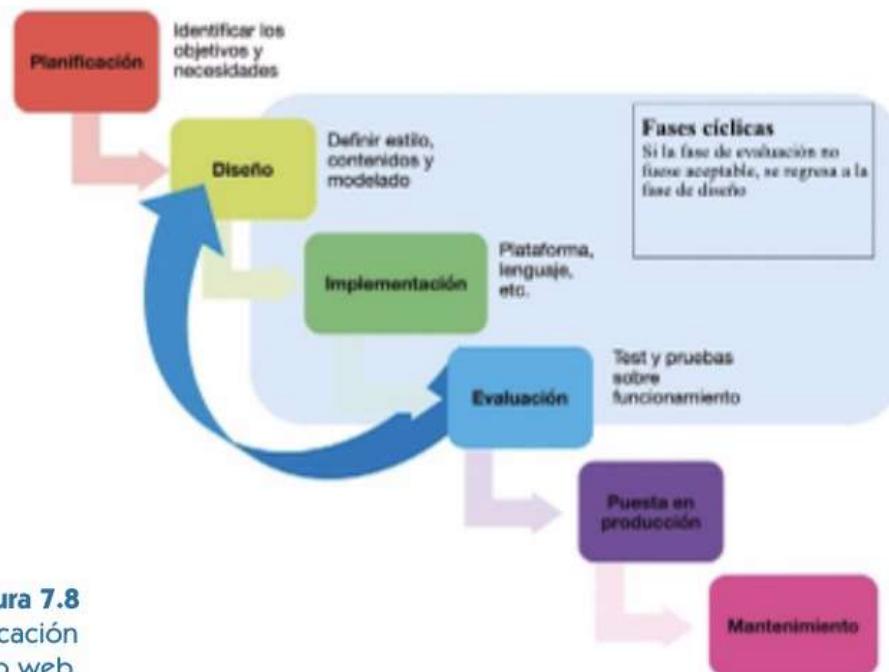
#### Actividad propuesta 7.4



¿Consideras que es mejor el método por inspección al estar basado en expertos profesionales en usabilidad? O, por el contrario, ¿consideras que el método de test a usuarios es más efectivo, puesto que se lleva a cabo sobre los receptores directos de una aplicación? Razona tu respuesta.

#### 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 7.8**  
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.

#### RECUERDA

- ✓ 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

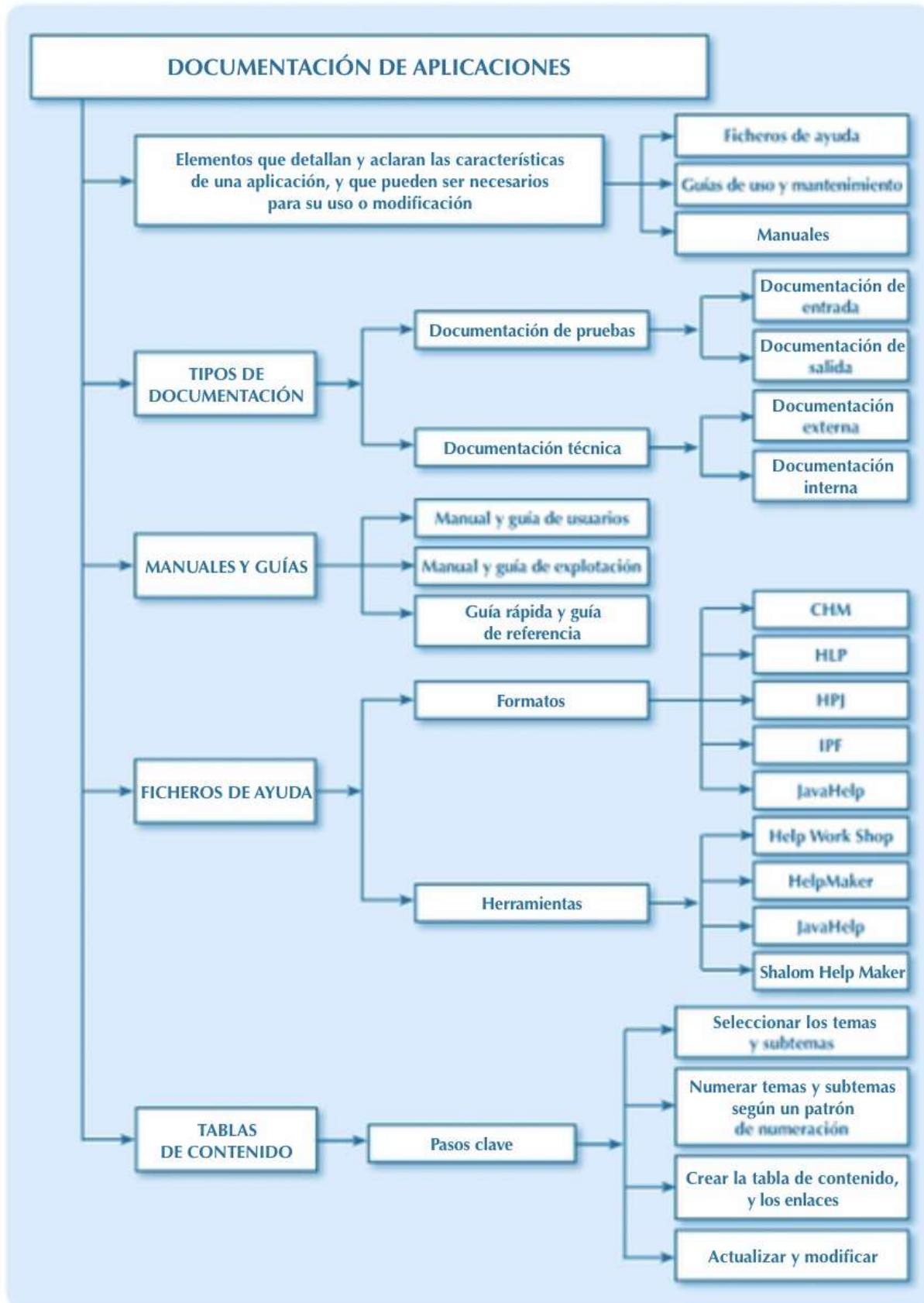
Se trata de la última fase del proceso, aunque debe tratarse como una fase crítica. Se trata de una fase fundamental, debido a que la tecnología evoluciona continuamente, y con ella las necesidades y posibilidades de los usuarios, por lo que es muy importante adaptar y mejorar el sitio a partir de esto.

# Documentación de aplicaciones

## Objetivos

- ✓ Conocer los diferentes sistemas de generación de ayudas que existen.
- ✓ Saber cuáles son los procedimientos para generar ayudas en los formatos habituales y de manera sensible al contexto.
- ✓ Comprender la importancia de la correcta documentación de aplicaciones.
- ✓ Realizar manuales de usuario y guías de referencia, así como manuales de instalación, configuración y administración, identificando claramente las diferencias existentes entre los mismos.

## Mapa conceptual



## Glosario

**Documentación de pruebas.** Documentación que incluye información relacionada con los requisitos y procedimientos de los escenarios de pruebas, así como de los informes resultantes de su realización.

**Documentación.** En el ámbito de las aplicaciones software, nos referimos a todos los elementos que detallan y aclaran las características de una aplicación y que pueden ser necesarios para su uso o modificación.

**Documentación técnica.** Contiene información de carácter técnico relacionada con una aplicación concreta, en la que se detallan las especificaciones y requisitos del sistema, guía de uso, etc. Está formada habitualmente por guías, hojas de especificaciones, manuales, etc.

**Fichero de ayuda.** Documento que contiene información de ayuda sobre un determinado campo. Un ejemplo de fichero de ayuda podría ser un manual de uso de una herramienta concreta, dirigido a los usuarios de la misma.

**Guía de referencia.** Documentación que está enfocada a usuarios que tienen un mayor nivel de conocimiento sobre el uso de la aplicación, así como una mayor experiencia.

**Guía rápida.** Las guías rápidas aportan información muy concreta, relacionada con diferentes procedimientos o campos de una aplicación.

**JavaHelp.** Es a la vez una herramienta para generar ficheros de ayuda y un fichero de ayuda en sí mismo. Está orientado a la documentación de aplicaciones desarrolladas en Java.

**Manual de usuario.** Contiene toda la información necesaria con el fin de hacer más fácil al usuario el uso y comprensión del programa. Los ámbitos de aplicación son varios y diversos: formación del usuario, guía de consulta ante dudas, ayuda para detectar y corregir errores.

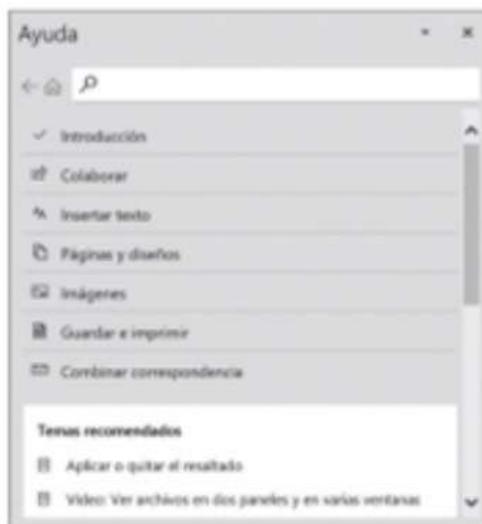
### 8.1. ¿Por qué es importante documentar?

Cuando se habla de documentación de aplicaciones software nos referimos a todos los elementos que detallan y aclaran las características de una aplicación, y que pueden ser necesarios para su uso o modificación. En concreto, se trata de ficheros de ayuda, manuales y demás guías de uso y/o mantenimiento, ya sean dirigidas para los usuarios de la aplicación, para equipos de soporte o incluso para otros diseñadores.

Para un sistema software es especialmente importante contar con una documentación completa y que sea sencilla de consultar. Los motivos para ello son varios:

- Facilitar su comprensión y posibilitar su uso por parte del usuario, independientemente de cuál sea su perfil.
- Facilitar el mantenimiento de la aplicación.

- Facilitar el desarrollo de mejoras por otros equipos de trabajo.
- Facilitar que el sistema pueda ser comprendido por otros diseñadores.



**Figura 8.1**  
Ejemplo de menú de ayuda  
de Microsoft Word.

A pesar de tratarse de un trabajo que puede resultar tedioso, se le debe otorgar toda la importancia que tiene. No es suficiente con desarrollar aplicaciones funcionales y atractivas, sino que, además, estas deben contar con una documentación adecuada. Se trata de una tarea tan importante como la de realizar el propio código del programa.

### Actividad propuesta 8.1



En tu opinión, ¿crees que la documentación de aplicaciones informáticas puede ser una tarea importante para el éxito de las mismas?

## 8.2. Tipos de documentación

A grandes rasgos, podemos dividir la documentación de aplicaciones software en dos grupos: documentación de pruebas y documentación técnica.

### 8.2.1. Documentación de pruebas

Documentar las pruebas realizadas sobre un programa determinado es fundamental para detectar y corregir posibles errores. Podemos distinguir a su vez dos tipos:

- a) *Documentación de entrada:* en la que se especifican los escenarios de prueba y se detallan los procedimientos de las mismas.
- b) *Documentación de salida:* se trata de los informes resultantes de aplicar las pruebas sobre el programa definidas en el punto anterior.

Estos informes recogerán el histórico de pruebas realizadas, documentándose así cada problema que haya podido surgir y su posterior corrección.

### 8.2.2. Documentación técnica

Pertenece a este grupo el resto de documentación: guías, hojas de especificaciones, manuales. Al igual que en el caso anterior, puede dividirse en dos grupos:

- a) *Documentación interna:* se trata de los comentarios incluidos por el desarrollador en el código, que deben describir distintos aspectos sobre el mismo.

Se debe incidir en la importancia de realizar un programa ordenado y claro, en el que los comentarios ayuden a entender el código, pero este ya de por sí debe ser claro. En este ámbito, debe tenerse en cuenta lo siguiente:

- Incluir comentarios al comienzo de módulos.
  - Comentar variables, contestantes, procedimientos y funciones.
  - Incluir comentarios introductorios sobre bloques de código, funciones o módulos.
  - No comentar lo obvio, ya que el exceso de comentarios puede ser contraproducente.
- b) *Documentación externa:* en este caso, suele tratarse de un manual técnico (orientado a programadores para facilitar el mantenimiento y desarrollo de la aplicación a futuro) y de un manual y/o guía de usuario (para facilitar su uso).



#### Actividad propuesta 8.2

¿Podrías indicar algún ejemplo que conozcas de cada uno de los tipos de documentación indicados anteriormente?

### 8.3. Manuales y guías

Existen diferentes tipos de manuales y guías, diferentes entre sí en función del contenido de cada uno y de su manera de utilización. Estos documentos pueden crearse en diferentes formatos y/o soportes: papel, formato electrónico, web de ayuda, presentación, vídeo, combinación de varios, según cuál sea la complejidad de la aplicación resultante y del perfil de usuario a quien vaya dirigido.

### 8.3.2. Manual y guía de explotación

El *manual y guía de explotación* contiene la información necesaria para utilizar y explotar la aplicación. Es decir, va muy orientado a la instalación, configuración y puesta en marcha, por lo que, evidentemente, irá muy ligado al contexto en el que se vaya a realizar la instalación (tipo de organización, requerimientos, número de usuarios).

Aunque en este caso tampoco hay una norma para su elaboración, debe contener al menos los siguientes puntos:

- ✓ Requerimientos mínimos del sistema en cuanto a procesador, memoria, espacio libre.
- ✓ Proceso de instalación:
  - Necesidad o no de preparación previa, si es necesaria alguna tarea para ello (por ejemplo, habilitar la conexión a Internet, activar o desactivar permisos).
  - Opciones de instalación: a través de USB, red, etc.
  - Procedimiento de instalación paso a paso, detallando y explicando sus diferentes opciones.
- ✓ Actualizaciones del sistema y copias de seguridad.
- ✓ Glosario.
- ✓ Índice.

En función de la complejidad de la aplicación, puede dividirse en manual de instalación y manual de configuración.

### 8.3.3. Guía rápida y guía de referencia

De manera adicional a los manuales estudiados anteriormente, la documentación de una aplicación puede completarse con este tipo de guías, en función del caso en concreto. Sus características son las siguientes:

1. *Guía rápida*: se orienta a usuarios del sistema y/o encargados de mantenimiento. En función de la complejidad de la aplicación puede ser necesario realizar varias, cada una con distinta temática. Las guías rápidas aportan información muy concreta y muy diferente, relacionada con diversos procedimientos o campos de una aplicación.
2. *Guía de referencia*: al contrario que en el caso de las guías rápidas, estas suelen estar pensadas para usuarios que tienen un mayor nivel de conocimiento sobre el uso de la aplicación, así como una mayor experiencia. Por ello, es habitual que contengan información relacionada con aspectos más técnicos: tipos de mensajes de error y su causa, tipos de datos de entrada que le son permitidos a la aplicación, tipos de comandos, etc.



#### Actividad propuesta 8.3

Obtén información y ejemplos sobre diferentes guías rápidas y de referencia de algunas aplicaciones que conozcas.



### TOMA NOTA

En el momento de realizar un manual o guía es muy importante pensar como lo haría el usuario que utilice la aplicación para entender qué problemas puede encontrarse y de qué forma podemos ayudarle a resolverlos. Este es el fin principal de los manuales y las guías de usuario.

### 8.3.1. Manual y guía de usuario

El *manual de usuario* contiene toda la información necesaria con el fin de hacer más fácil el uso y la comprensión del programa. Los ámbitos de aplicación son varios y diversos: formación del usuario, guía de consulta antes dudas, ayuda para detectar y corregir errores, etc.

Es cierto que no existe ninguna norma acerca de cómo se debe elaborar, aunque hay ciertos patrones y usos que se han de seguir. En primer lugar, el documento en sí mismo debe ser claro y atractivo desde el punto de vista del usuario, que a la postre es quien va a utilizarlo. Para ello, puede resultar aconsejable utilizar imágenes y gráficos para ayudar a entender el texto, utilizar un vocabulario claro y conciso, ejemplos prácticos, etc.

Una posible estructura del manual de usuario puede ser la siguiente:



**Figura 8.2**

Possible estructura de manual de usuario de una aplicación.

### RECUERDA

- ✓ La estructura de un manual de usuario dependerá en gran medida del objetivo que se pretenda conseguir, así como del tipo de usuario que vaya a utilizarlo.

## 8.4. Ficheros de ayuda. Formatos. Herramientas para generarlos

Un *fichero* es un elemento que puede contener varios tipos de información en diferentes formatos, ya sea en soporte físico o en soporte digital, que suele ser el procedimiento habitual en el campo de las aplicaciones informáticas.

De esta forma, un *fichero de ayuda* es un documento que contiene información de ayuda sobre un determinado campo. Un ejemplo de fichero de ayuda podría ser un manual de uso de una herramienta concreta, dirigido a los usuarios de la misma. Los ficheros de ayuda están formados por dos partes diferenciadas:

- *Mapa del fichero*: se trata de un apartado que facilita la navegación por el fichero, que identifica claramente el contenido de este a través de identificadores.
- *Vista de información*: muestra al usuario contenidos en forma de glosario o tabla de contenidos.

### 8.4.1. Formatos

En el siguiente cuadro se muestran algunos de los principales formatos de ficheros de ayuda, junto con las principales características de estos:

**CUADRO 8.1**

Comparativa de algunos de los principales formatos de ficheros de ayuda existentes

Formato	Características	Plataforma
CHM	<ul style="list-style-type: none"> <li>– Generado a partir de HLP.</li> <li>– Utiliza HTML para generar la ayuda.</li> <li>– Enlaces mediante hipervínculos a la tabla de contenido.</li> <li>– Permite fusionar varios ficheros de ayuda.</li> <li>– Puede ser creado a partir de HTML Help Workshop.</li> </ul>	Microsoft Windows
HLP	<ul style="list-style-type: none"> <li>– Puede incluir tabla de contenido en fichero .cnt.</li> <li>– Incluye información extra en fichero .gid.</li> <li>– Utiliza ficheros RTF para generar la ayuda.</li> <li>– Necesita compilación (por ejemplo mediante HTML Help Workshop).</li> </ul>	Windows
HPJ	<ul style="list-style-type: none"> <li>– Contiene tabla de contenido (fichero .cnt).</li> <li>– Se crea mediante herramienta tipo Help Workshop.</li> </ul>	Windows
IPF	<ul style="list-style-type: none"> <li>– Similar a HTML.</li> <li>– Se utiliza para ayuda en línea o web.</li> <li>– Es necesario compilar para generarla.</li> </ul>	Web (ayuda en línea)
JavaHelp	<ul style="list-style-type: none"> <li>– Implementado en Java, por lo que es utilizado habitualmente para implementar la ayuda de aplicaciones desarrolladas en este lenguaje.</li> </ul>	Varias

## RECUERDA

- ✓ La elección del formato depende de cómo se implemente la ayuda de la aplicación en sí misma. Es importante tener claro este aspecto previamente a la realización del contenido para intentar optimizarlo.

#### 8.4.2. Herramientas para generar ficheros de ayuda

A continuación, se detallan las principales características de algunas herramientas que se pueden utilizar para generar ficheros de ayuda:

- ✓ *Help Work Shop*: posibilita crear ficheros de ayuda en entornos Windows. Es una aplicación que consta de un editor de imágenes, un administrador de proyectos y un compilador.
- ✓ *HELPMAKER*: es una herramienta de carácter gratuito que cuenta con múltiples opciones para personalizar los ficheros. Utiliza un editor de texto para facilitar la generación de los ficheros de ayuda.
- ✓ *JavaHelp*: JavaHelp por sí mismo es un sistema de creación de ficheros de ayuda y, además, un formato de ficheros de ayuda propiamente dicho. Se caracteriza por:
  - Utilizar Java para su implementación.
  - Utilizarse en aplicaciones Java.
  - Ser independiente de la plataforma.
  - Facilitar la realización de ayuda online.
- ✓ *SHALOM HELP MAKER*: se trata de otra herramienta gratuita, que utiliza un editor de texto para la creación de ficheros de ayuda, de manera similar a HELPMAKER. Permite incluir enlaces externos, insertar o crear imágenes, crear índices, etc.

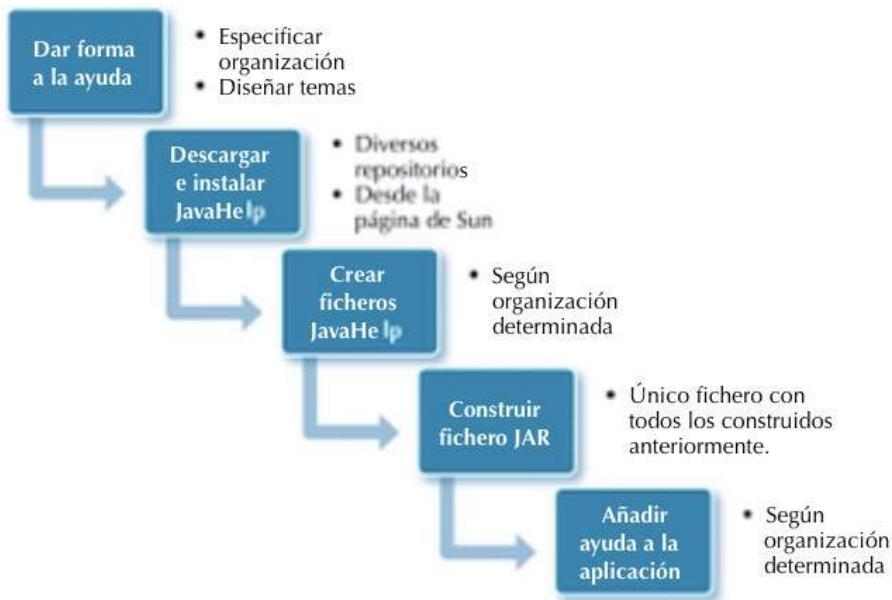


#### Actividad propuesta 8.4

Obtén información acerca de las herramientas más utilizadas y extendidas para generar ficheros de ayuda. Puedes emplear para ello buscadores vía web.

#### 8.4.3. Generación de un sistema de ayuda con JavaHelp

Para crear un sistema de ayuda con JavaHelp se deben seguir unos pasos concretos, indicados en el siguiente diagrama:



**Figura 8.3**  
Pasos necesarios para generar un sistema de ayuda con JavaHelp.

En el siguiente cuadro vemos los ficheros JavaHelp y para qué se utilizan:

**CUADRO 8.2**  
Ficheros JavaHelp y su uso

Fichero	Extensión	Utilización
Índice	XML	Incluye la distribución del sistema de ayuda.
Map	JHM	Asocia elementos (imágenes, ficheros HTML, etc.) del fichero HTML con un identificador.
Helpset	HS	Contiene la información necesaria para que el sistema de ayuda se ejecute.
Tabla de contenidos	XML	Incluye el contenido de la ayuda y su distribución.
Temas o topics	HTML	Para crearlos se puede utilizar cualquier herramienta para generar HTML. Contienen la información de ayuda como tal, debiéndose realizar uno por cada tema o asunto de ayuda. Se muestra uno de ellos por pantalla. Deben tener organización jerárquica.
Base de datos de búsqueda	N/A	Se debe utilizar la herramienta jhindexer para generarla.

El directorio con los ficheros creados debe seguir una estructura similar a la siguiente:

```

/help
  - Índice.xml
  - Map.jhm
  - Helpset.hs
  - TablaDeContenidos.xml
/html
  - Tema1.html
  - Tema2.html
/JavaHelpSearch

```

**Figura 8.4**  
Directorio de ficheros JavaHelp.

Es fundamental que los ficheros HTML estén en el directorio ./help, ya que, en caso contrario, la aplicación no funcionará.

Para generar fichero JavaHelp de prueba vamos a seguir paso a paso lo indicado anteriormente. Para visualizar un ejemplo, nos crearemos una ayuda ficticia basada en tres páginas HTML:

- Página principal a modo de introducción.
- Página con la ayuda necesaria para el “tema 1”.
- Página con la ayuda necesaria para el “tema 2”.

El resultado final será el siguiente:



**Figura 8.5**  
Ejemplo de ayuda creada con JavaHelp.

Veamos el procedimiento paso a paso.

### 1. Dar forma a la ayuda

En este primer paso, construiremos la ayuda de nuestra aplicación, creando tantos ficheros HTML como necesitemos. Es importante construir adecuadamente la estructura de la ayuda,

ya que es fundamental que sea clara y concisa para facilitar la localización de la información por parte del usuario.

En nuestro ejemplo, al tratarse de un caso sencillo, únicamente hemos creado dos temas de ayuda, junto a una página principal.

El código HTML del fichero principal es el siguiente:

```
<!DOCTYPE html>
<html>
<head>
<meta charset=utf-8 />
<title>Esta es nuestra ayuda JavaHelp de prueba </title>
</head>
<body>

<h1>Esta es una ayuda creada con JavaHelp</h1>

<h2>Esta ayuda la hemos creado a modo de prueba</h2>

<p>Para ello podemos utilizar un generador de código HTML cualquiera</h2></h2>
<p>Podemos también añadir imágenes o incluso referenciar
entre sí las diferentes páginas de la ayuda</h2></h2>



<p></p>

Este es el enlace al

<a href="tema1.html">Tema 1</a> de ayuda

<p><a>Este es el enlace al </a>
    <a href="tema2.html">Tema 2</a> de ayuda

</body>
</html>
```

## 2. Descargar e instalar JavaHelp

Los ficheros necesarios para poder trabajar con JavaHelp los podemos encontrar en diversos repositorios. Es recomendable descargarlos desde un sitio seguro y oficial, como puede ser la página de Sun Microsystems.

Una vez descargados, seguiremos el proceso de instalación hasta completarlo.

### 3. Crear ficheros JavaHelp

- *Índice.* Será un fichero con extensión .xml en el que se detallarán todas las páginas que tiene nuestra ayuda (tres en nuestro ejemplo). El fichero tendrá la siguiente apariencia: se compone de una etiqueta principal llamada <index>, que a su vez contiene varias subetiquetas llamadas <indexitem>, que hacen referencia a cada elemento de ayuda que compondrá el Índice. De esta forma, se mostrará el texto que se incluya en “text”, mientras que el campo “target” hace referencia a la clave o ID. Esta clave debe coincidir con la indicada en el fichero map dentro del mismo campo “target”.
- *Fichero map.* En este caso, hemos creado un “mapID” por cada fichero HTML, con el fin de poder referenciarlo. Evidentemente, se debe indicar la ruta en la que está dicho fichero. El fichero se puede crear con cualquier editor de texto, salvándolo en formato .jhm.
- *Helpset.* Se podría decir que es el fichero principal, que debe construirse en formato .xml, aunque se guardará con extensión .hs.

Un ejemplo de fichero HelpSet para nuestro caso puede ser el siguiente, aunque habría diversas posibilidades: el contenido del fichero se encuentra dentro de la etiqueta <helpset> que define el principio y final del mismo. A partir del comienzo, podemos definir tres tipos de etiquetas:

<title>: será el título que aparezca en la ventana de ayuda.  
 <maps>: indica la ubicación del fichero map.  
 <homeID>: fichero que aparece por defecto al ejecutar la ayuda.  
 <mapref>: nombre del fichero map.  
 <view>: uno por cada elemento de la ayuda. En este caso, Búsqueda, Índice y Tabla de Contenidos.

El orden de los mismos en el fichero determina cómo aparecen en la ayuda:

<name>: nombre del view.  
 <label>: etiqueta que se mostrará en relación con este elemento.  
 <type>: muestra la clase de JavaHelp relacionada con el contenido.  
 <data>: indica el fichero que está relacionado con esa parte, de los que se crearon anteriormente.

En este caso, en el campo de Búsqueda, se incluye el subdirectorio que se crea a partir de la herramienta jhindexer.jar, como se verá a continuación.

- *Tabla de contenidos.* Detallará la organización de los ficheros de nuestra ayuda, que en nuestro ejemplo es muy sencilla. Deberá guardarse en formato .xml, con cualquier editor de texto. La etiqueta <toc> describe el elemento en su conjunto, que a su vez se divide en varios <tocitem>. Estos elementos pueden anidarse entre sí. Dentro de los mismos, de igual forma que en el fichero Índice, se mostrará el texto que se incluya en “text”. Así mismo, el “target” hace referencia a la clave o ID, que debe coincidir con la indicada en el fichero map.
- *Base de datos de búsqueda.* Para crearla es necesaria la ejecución del comando jhindexer.jar, que invoca a una de las herramientas de JavaHelp.

Para ello, basta con ejecutar la orden “jhindexer.jar html”, cambiando previamente el PATH al directorio en el que tenemos los ficheros de ayuda, es decir, el directorio

help. El parámetro html hace referencia al directorio en el que están almacenados los ficheros de ayuda.

La creación de esta base de datos permitirá contar con una pestaña de búsqueda, que hará posible navegar por la ayuda de una manera más ágil y eficaz.

#### 4. Construir fichero JAR

Para construir los ficheros JAR será necesario ejecutar: Jar -cvf help.jar. Mediante esta orden se crean ficheros JAR independientes. Se debe ejecutar desde el directorio help en el que están los ficheros de ayuda.

#### 5. Añadir ayuda a la aplicación

Para añadir la ayuda que hemos creado a cualquier aplicación Java, se deben incluir los siguientes apartados en el mismo:

Paquete javax.help.\*

Clase HelpSet, que posibilita el uso de ficheros y datos de la ayuda.

Clase HelpBroker, que ayuda a visualizar el contenido de la ayuda.

Método createHelpBroker.

Método findHelpSet.

Método enableHelpOnButton, para poder visualizar la ayuda al pulsar un menú.

Método enableHelp, que al hacer clic sobre un elemento indicará el tema de ayuda.

Método enableHelpKey, que activa la tecla de ayuda sobre un elemento.

#### 8.4.4. Ayuda genérica y sensible al contexto

Aunque a priori pudiera parecer una distinción evidente, debemos tener en cuenta las diversas formas en las que puede presentarse y/o elaborar una documentación de ayuda relacionada con una aplicación:

#### CUADRO 8.3

#### Características de los diferentes tipos de ayuda

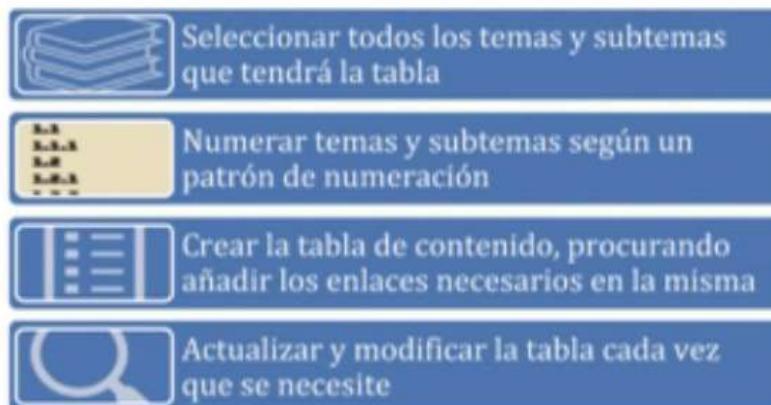
Ayuda genérica	Ayuda sensible al contexto
<ul style="list-style-type: none"> <li>- Contiene toda la información de ayuda sobre una determinada aplicación, dividida por temas o campos.</li> <li>- Generalmente presenta un índice o menú.</li> <li>- Permite navegar o buscar entre los elementos que la forman.</li> </ul>	<ul style="list-style-type: none"> <li>- Se trata de ayuda particularizada para un tema o acción concreta.</li> <li>- Puede presentarse de diferentes maneras: ayuda en línea, breve detalle del uso de un menú en el programa.</li> </ul>

## 8.5. Tablas de contenidos

Las tablas de contenidos nos ayudan a modelar tanto el contenido como la estructura de un documento determinado. La información estará organizada en forma de esquema, con diversos niveles entre los que se cabe destacar títulos y subtítulos. Lógicamente, el diseño de las mismas depende de quién haga el desarrollo, así como de otros muchos factores (objetivo, usuario al que se dirigen, etc.). Sin embargo, existe una serie de características comunes a todas ellas:

- ✓ Es posible o no que muestren el número de páginas que contienen, depende del diseño que se realice sobre las mismas. Se trata de un elemento completamente opcional.
- ✓ Las tablas de contenidos habitualmente contienen enlaces directos a cada título o subtítulo, que apunta al contenido de los mismos. Así se consigue una navegación y utilización más fluida y sencilla para el usuario.
- ✓ Suelen situarse al comienzo de cada documento, de manera similar al índice de los libros.
- ✓ Aunque puede resultar sencillo, su proceso de elaboración no es elemental, ya que es necesario realizar un análisis previo y pormenorizado de toda la documentación que contienen. Además, es necesario que los títulos sean lo más claros e intuitivos posible. Es también importante que no haya información duplicada en varias partes del documento.

Como resumen, en la siguiente figura se muestran algunos consejos que se han de tener en cuenta para elaborar una tabla de contenidos clara y de calidad, que sea útil para el usuario:



**Figura 8.6**

Consejos para elaborar una tabla de contenidos de calidad.



### Actividad propuesta 8.5

¿Cuáles son las principales características que debe cumplir una tabla de contenidos? ¿Podrías indicar algún ejemplo real que conozcas?

## Resumen

- Tal y como hemos estudiado en este capítulo, es muy importante documentar de forma adecuada cada aplicación o programa que se realice. Es decir, el desarrollo de cualquier aplicación (también el desarrollo de interfaces gráficas) implica la elaboración de manuales y sistemas de ayuda al usuario, de modo que se establezca una documentación completa sobre el funcionamiento de la misma. Estos sistemas de ayuda resultan clave tanto para un usuario básico como para usuarios con mayor experiencia, y pueden ayudar al éxito de la aplicación en sí.
- Para lograr esa tarea de elaborar una documentación completa y atractiva, se pueden utilizar diferentes tipos de documentaciones, que se pueden adaptar a cada caso concreto y a cada aplicación. A grandes rasgos, podemos diferenciar un primer tipo de documentación denominada “de pruebas” de otro tipo de documentación denominada “técnica”. La documentación técnica quedará dividida a su vez en documentación de entrada y documentación de salida, mientras que la de tipo técnico se subdivide en documentación interna y documentación externa.
- La documentación de ayuda suele estar formada por *ficheros de ayuda*. Un ejemplo de fichero de ayuda podría ser un manual de uso de una herramienta concreta, dirigido a los usuarios de la misma. Los ficheros de ayuda están formados por el mapa del fichero y por la vista de información.
- En cuanto a los formatos de los ficheros de ayuda, algunos de los más comunes son los siguientes: CHM, HLP, HPJ, IPF y JavaHelp.
- Existen multitud de herramientas que nos ayudan a generar ficheros de ayuda, como pueden ser Help Work Shop, HELPMAKER o JavaHelp. En el caso de JavaHelp, es una aplicación muy utilizada para generar ayuda en aplicaciones desarrolladas en Java. Para su uso, deben seguirse varios pasos para la creación de un sistema de ayuda:
  1. Dar forma a la ayuda, indicando la organización y la temática.
  2. Descargar e instalar el entorno JavaHelp.
  3. Creación de los ficheros necesarios para el uso de JavaHelp.
  4. Construir un fichero JAR utilizando estos ficheros.
  5. Añadir la ayuda a la aplicación.
- Los ficheros que se utilizan en JavaHelp son los siguientes:
  - Índice.
  - Map.
  - HelpSet.
  - Tabla de contenidos.
  - Temas o topics.
  - Base de datos de búsqueda.
- Por último, las tablas de contenido permiten indicar la estructura y el contenido de un documento determinado. Habitualmente, están estructuradas en diferentes niveles de elementos, entre los que cabe destacar los títulos y los subtítulos.

## Ejercicios propuestos



- Como hemos visto en este capítulo, un sistema de ayuda sirve para detallar las características de una aplicación e indicar qué elementos pueden ser necesarios para su uso, mantenimiento o modificación. La clave se encuentra en realizar una documentación completa y que a la vez sea sencilla de consultar.

En este caso se utilizará la herramienta JavaHelp para la inclusión de un sistema de ayuda. Esta aplicación permite la creación de sistemas de ayuda para ser integrados posteriormente en un programa desarrollado en lenguaje Java. Será necesario elaborar cuatro ficheros para desplegar el sistema de ayuda: map\_file.jhm, toc.xml, index.xml y help\_set.hs.

- Primero se debe especificar la organización y diseñar los temas de la ayuda.
- Después será necesario instalar JavaHelp.
- Crear los ficheros de JavaHelp necesarios. En este caso será necesario implementar cuatro ficheros llamados:
  - Fichero 1: map\_file.jhm
  - Fichero 2: toc.xml
  - Fichero 3: index.xml
  - Fichero 4: help\_set.hs

Los tres primeros ficheros son los referenciados desde el archivo hep\_set.hs. Este documento relaciona todo el sistema de ayuda, incluyendo el índice y la tabla de contenidos.

Vamos a construir los ficheros JavaHelp necesarios para documentar una aplicación llamada "Calculadora", que se encarga de realizar las cuatro operaciones básicas: suma, resta, multiplicación y división.

- Continuando con los pasos del ejercicio 1, construye un fichero JAR que incluya todos los ficheros y permita mejorar su distribución. Después añade la ayuda a la aplicación "Calculadora" desarrollada en temas anteriores.
- Si tuvieras que diseñar una guía / manual de usuario de esta aplicación "Calculadora", ¿cómo lo construirías? ¿Qué puntos contendría?

## ACTIVIDADES DE AUTOEVALUACIÓN

- La parte del fichero de ayuda que se muestra al usuario, habitualmente en forma de índice, glosario, tabla de contenido e incluso buscador de temas, es:
  - a) La guía de inicio.
  - b) El mapa de navegación.
  - c) El mapa de fichero.
  - d) La vista de información.

2. Este tipo de formato de fichero de ayuda actualmente se encuentra en desuso:
- a) HPJ.
  - b) CHM.
  - c) HLP.
  - d) IPF.
3. Este tipo de formato de fichero de ayuda incluía ciertas mejoras sobre HPL:
- a) CHM.
  - b) HPJ.
  - c) IPF.
  - d) JavaHelp.
4. Este tipo de formato de fichero de ayuda utiliza para la representación de gráficos ficheros de tipo .shg:
- a) HPJ.
  - b) CHM.
  - c) IPF.
  - d) JavaHelp.
5. La herramienta Shalom Help Maker:
- a) Permite la creación de ficheros de ayuda para Windows y tras su descarga es necesario ejecutar el software htmlhelp.exe para instalarla.
  - b) Toda la información queda contenida en un único fichero, por lo que es posible exportar toda la ayuda en un solo documento PDF.
  - c) Es una herramienta de pago.
  - d) Permite creación de índices, contenidos enlazados a otras páginas, links externos, así como la creación de imágenes que enriquecen el contenido del documento final.
6. Indica cuál de las siguientes afirmaciones es falsa respecto a las tablas de contenidos:
- a) Las tablas de contenido se sitúan al final de cualquier documento, como si de un índice de libro se tratara.
  - b) Pueden mostrar el número de página o no, en función del diseño.
  - c) Casi siempre incluyen un enlace directo en cada uno de los títulos o subtítulos por los que aparecen compuestas.
  - d) No debe duplicarse la información.
7. La documentación técnica:
- a) Sirve para documentar las pruebas realizadas sobre un programa determinado.
  - b) Se trata de los informes resultantes de aplicar las pruebas sobre los programas.
  - c) Es la documentación en la que especifican los escenarios de prueba y se detallan los procedimientos de estas.
  - d) Son las guías, hojas de especificaciones, manuales, etc.
8. ¿Qué tipo de manual o guía contiene la información necesaria para poner en uso una aplicación?:
- a) El manual de usuario.
  - b) El manual de explotación.
  - c) La guía rápida.
  - d) La guía de referencia.

9. JavaHelp requiere del uso de diferentes ficheros y de diferentes tipos de estos para el diseño de cada una de las partes que forman la ayuda final, como el fichero Helpset que:

- a) Incluye la distribución del sistema de ayuda.
- b) Asocia elementos del fichero HTML con un identificador.
- c) Contiene la información necesaria para que el sistema de ayuda se ejecute.
- d) Contiene la información de ayuda como tal, debiéndose realizar uno por cada tema.

10. La clase HelpBroker permite visualizar el contenido de la ayuda desde la aplicación, ¿cuál de los siguientes no es uno de sus principales métodos?:

- a) enableHelpKey.
- b) enableHelpOnButton.
- c) enableHelp.
- d) findHelpSet.

#### SOLUCIONES:

1.  a  b  c  d

2.  a  b  c  d

3.  a  b  c  d

4.  a  b  c  d

5.  a  b  c  d

6.  a  b  c  d

7.  a  b  c  d

8.  a  b  c  d

9.  a  b  c  d

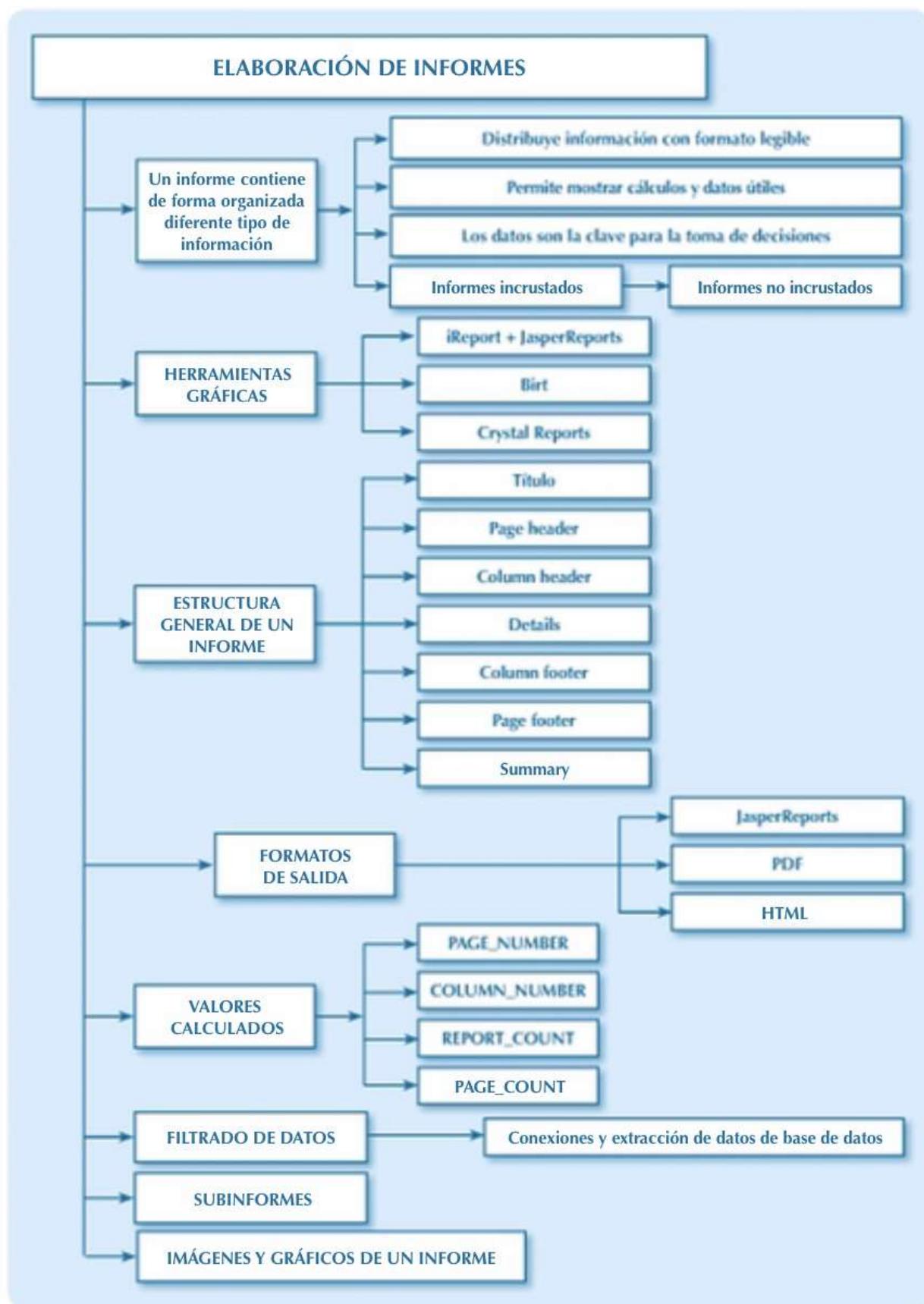
10.  a  b  c  d

# Elaboración de informes

## 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 existentes 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.

**Sección.** Una de las partes en las que se divide un informe: título, encabezado, contenido, pie, etc.

**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.

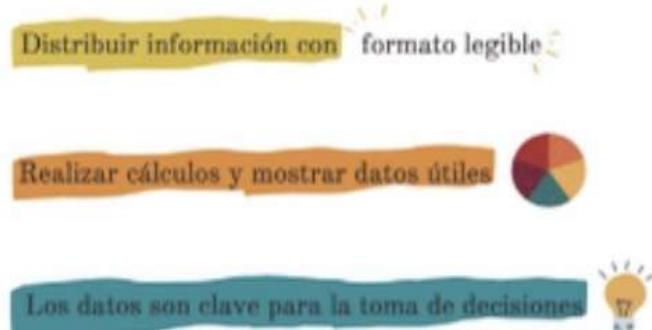
### 9.1. Informes en la aplicación

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.

Existen diferentes tipos de informes, en función de cómo estén vinculados con la aplicación. Podemos distinguir los informes incrustados de los informes no incrustados.

#### RECUERDA

- ✓ 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.

### 9.1.1. 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.

### 9.1.2. 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 crean a partir de aplicaciones específicas para ello, que no están integradas con la aplicación concreta ni con su entorno de desarrollo.



### Actividad propuesta 9.1

¿Conoces alguna aplicación que contenga un módulo para obtener informes?  
¿Cuál es y cómo se utiliza? ¿Conoces alguna aplicación que permita obtener informes no incrustados?

## 9.2. 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.

#### TOMA 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.



### Actividad propuesta 9.2

¿Conoces algún otro ejemplo de herramienta gráfica para la generación de informes?

### 9.3. 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 9.2**  
Plantilla de informe con las diferentes secciones en JasperReports.

#### RECUERDA

- ✓ 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).

#### Actividad propuesta 9.3



En tu opinión, ¿crees que son indispensables todas las secciones analizadas anteriormente para confeccionar cualquier tipo de informe?

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

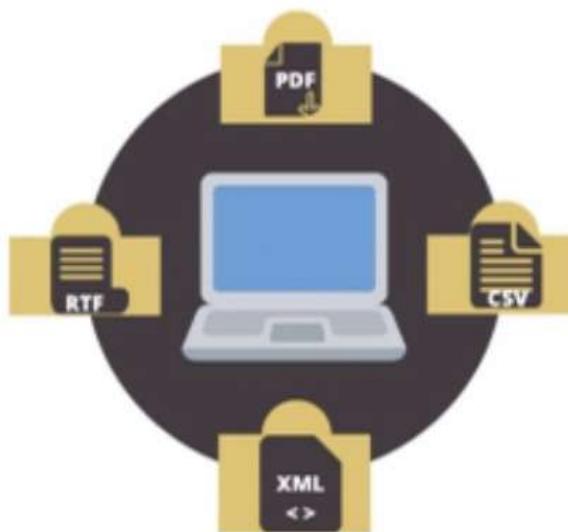
**CUADRO 9.1**  
Secciones habituales en un informe

<i>Título</i>	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.
<i>Page Header</i>	Contiene datos generales sobre el informe, como el autor, la fecha de generación.
<i>Column Header</i>	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.
<i>Details</i>	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.
<i>Column footer</i>	Se utilizan habitualmente para incluir datos relacionados con el contenido de cada columna del informe.
<i>Page footer</i>	De manera similar al encabezado, contiene datos generales sobre el informe, como pueden ser los números de página, por ejemplo.
<i>Summary</i>	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.

#### 9.4. 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.

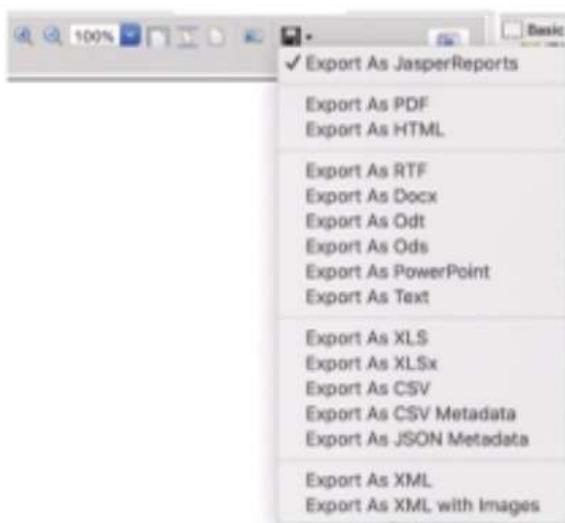
La gran cantidad de formatos posibles que soportan las herramientas para la creación de informes es una de sus principales ventajas, ya que aporta un gran abanico de opciones que se adaptan a prácticamente cualquier necesidad, posibilitando una adaptación del formato de salida al objetivo que se desea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



**Figura 9.3**  
Formatos de salida más utilizados.

port 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 9.4:



**Figura 9.4**  
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.



#### TOMA 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.

## 9.5. 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:

- Variabes 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.
- Variabes 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 9.2**  
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).

### 9.5.1. Numeración de líneas. Recuentos y totales

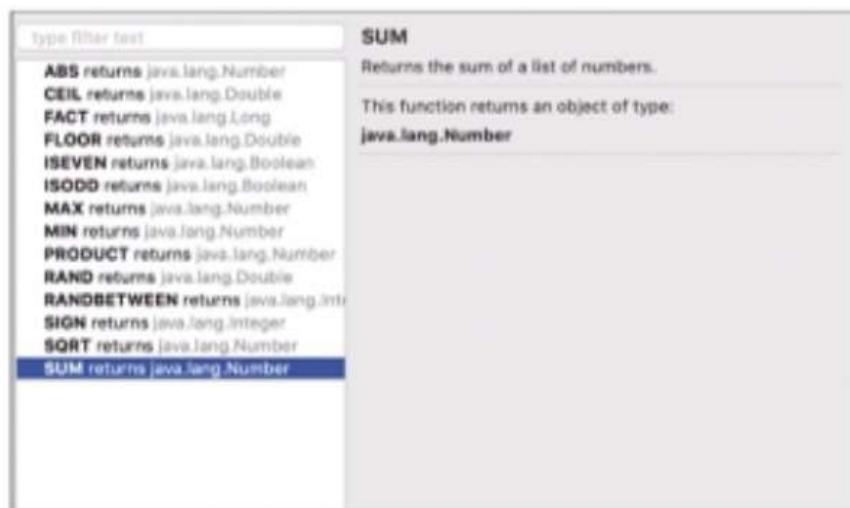
Los valores calculados se suelen utilizar a modo de resumen de los datos que contiene un informe. Por lo tanto, hay algunas operaciones comunes que se realizan con los mismos, como las siguientes:

- 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.



#### Actividad propuesta 9.4

¿Podrías indicar algunos ejemplos prácticos basados en casos reales en los que sea útil conocer la numeración de líneas o los recuentos y totales?



**Figura 9.5**  
Operaciones más utilizadas sobre columnas

## 9.6. 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.

### 9.6.1. 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 <condición> [AND | OR <condición>]]
[GROUP BY <nombre_campo>]
[HAVING <criterios_de_agrupación>]
[ORDER BY <nombre_campo> | <índice_campo>
[ASC|DESC]];
```

**Figura 9.6**  
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 cuales 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 9.7**  
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}
```

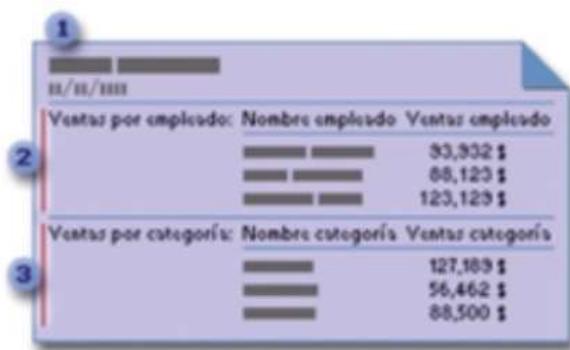
**Figura 9.8**  
Ejemplo de consulta aplicando filtrado de datos.

## 9.7. 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 9.9**  
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 9.3**  
Diferencias entre informes y subinformes

Informes	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

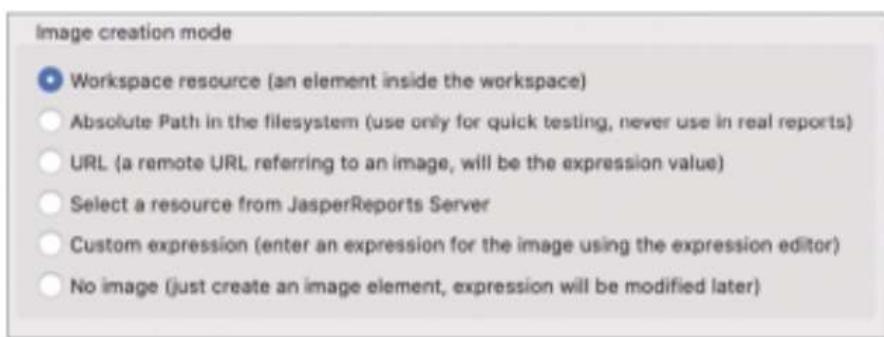
## 9.8. Imágenes y gráficos en un informe

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.

### 9.8.1. 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 9.10**  
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.

### 9.8.2. Inclusión de gráficos

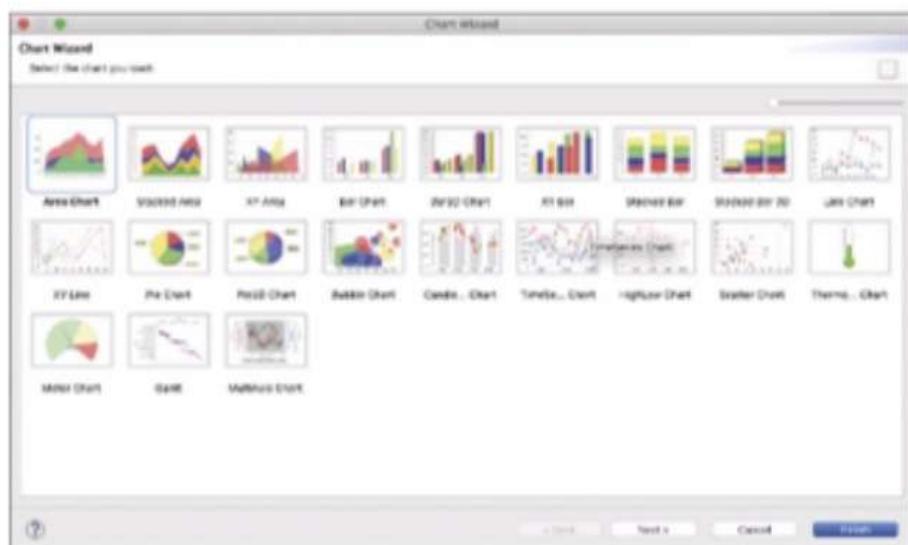
La inclusión de gráficos en un informe debe ser un hecho cotidiano y habitual, siempre que se trabaje con datos cuantificables. Aportan mucha claridad al informe, ya que permiten mostrar visualmente los mismos datos que puede contener el informe en forma de datos o cifras.

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 líneas 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.

### Actividad propuesta 9.5



¿Qué tipo de gráfico utilizarías en un informe de ventas? ¿Y en un resumen sobre la evolución de un proyecto de desarrollo de una aplicación informática, en cuanto a plazos y en cuanto a costes?

### Resumen

- Los informes obtenidos de una aplicación permiten extraer y analizar diferente tipo de información, desde datos relativos al uso de la aplicación hasta otro tipo de contenido que permitan analizar los datos tomados desde la aplicación en su conexión a un origen de datos.
- Para su elaboración, existen diversas herramientas que facilitan esta labor, como pueden ser JasperReports + iReports, especialmente utilizadas en la elaboración de informes en el ámbito del desarrollo de aplicaciones, de manera conjunta con herramientas de desarrollo como Eclipse.

- La creación de informes permite adaptar la presentación de los datos finales a múltiples tipos de escenarios, tanto en cuanto a las plantillas utilizadas como a la forma de mostrar la información, el manejo de estas herramientas vinculadas a entornos de desarrollo de programación son una herramienta clave. Así mismo, el formato de elaboración de los informes puede ser muy variado y la elección de uno u otro formato dependerá de múltiples factores: tipos de datos, objeto del informe, plataforma de aplicación, etc.
- Los informes obtenidos permiten utilizar elementos estáticos que se emplean sobre todo para definir la plantilla del informe, pero necesitan de la conexión a una fuente de datos para iterar sobre ella y mostrar los datos contenidos.
- Cuando se conecta un informe a una base de datos no siempre será necesario que se muestren todos los datos, campos o columnas, sino que en muchas ocasiones solo será necesaria parte de la información. Para conseguir esta selección se utilizará el modelado de consultas que permiten el filtrado de los datos.
- La elaboración de subinformes es muy importante para construir informes más robustos y completos. Las características de informes y subinformes son muy similares, pero también existen algunas diferencias entre ellos, dado que un informe puede contener subinformes, pero no a la inversa.
- Así mismo, la inclusión de imágenes y gráficos en un informe es habitualmente una buena práctica, dado que ayuda a la comprensión de los datos reflejados en el informe, de una manera sencilla, rápida y atractiva para el destinatario del informe.
- JasperReport incluye muchas herramientas y elementos que permiten la creación de complejos informes en los que se muestran todos los datos necesarios, permitiendo además la personalización de estos, desde la inclusión de imágenes y gráficos hasta la creación de subinformes o la edición de expresiones.

### Ejercicios propuestos



1. En este caso se realiza un diseño como el que sigue en el que se muestran todos los datos de productos (nombre y precio) de la tabla PRODUCT contenida en la base de datos Sample DB de JasperReports.

Se ha de guardar el fichero del informe en varios formatos que cumplan las siguientes especificaciones. Escoge y justifica el formato más adecuado para cada uno de los casos:

- a) Informe mostrado en línea al pulsar sobre un enlace en una página web.
- b) Se desea trabajar con los datos descargados utilizando herramientas de cálculo como Excel o Number.
- c) El fichero descargado se va a enviar como adjunto en un correo electrónico a los miembros de una junta directiva y este no se puede modificar.

2. Se pide generar un nuevo informe en JasperReports utilizando una plantilla de diseño en blanco para, a continuación, añadir elementos sobre el mismo. Se deben utilizar elementos básicos de la paleta, similares a los estudiados a lo largo del tema, poniendo en práctica las diferentes opciones de diseño que ofrece la herramienta.
3. Se pide generar un nuevo informe utilizando una plantilla de diseño en blanco y la base de datos de Sample DB incluida en JasperReports, que incluya los siguientes datos:

NOMBRE DEL PRODUCTO (Tabla Product)

PRECIO DEL PRODUCTO (Tabla Product)

El informe obtenido deberá enriquecerse con la inclusión de imágenes y gráficos.

## ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿Cuál de las siguientes afirmaciones es falsa en cuanto a la confección de informes para aplicaciones?:
  - a) Existen dos tipos de informes atendiendo a la forma en la que estos quedan vinculados con la aplicación: incrustados y no incrustados.
  - b) Un informe es un documento que recopila y muestra diferente tipo de información procedente de fuentes de datos a los usuarios, en función de las consultas realizadas.
  - c) La creación de los informes es estática.
  - d) Es posible seleccionar entre múltiples elementos y distribuciones de diseño.
2. ¿Cuál de las siguientes herramientas modela la encapsulación relativa a la generación de informes?:
  - a) Eclipse Pro.
  - b) XCode.
  - c) JasperReport.
  - d) iReport.
3. ¿Qué herramienta de código abierto permite la creación de informes de forma dinámica para entornos de inteligencia empresarial?:
  - a) JasperReport.
  - b) Eclipse Pro.
  - c) Eclipse Birt.
  - d) iReport.

4. ¿Qué afirmación es correcta en cuanto a las variables de usuario y variables predefinidas?:

- a) Las variables predefinidas suelen corresponder con el nombre de los campos del origen de datos vinculado al informe.
- b) Las variables de usuario no pueden corresponder con el nombre de los campos del origen de datos vinculado al informe.
- c) Las variables predefinidas aparecen por defecto en la herramienta, por lo que siempre van a estar presentes y modelan elementos habituales en la generación de cualquier informe.
- d) Las variables de usuario aparecen por defecto en la herramienta, por lo que siempre van a estar presentes y modelan elementos habituales en la generación de cualquier informe.

5. Qué tipo de operaciones se utilizan sobre los valores contenidos en una columna?:

- a) Recuentos y totales.
- b) Selecciones.
- c) Anulaciones.
- d) Ninguna de las respuestas anteriores es correcta.

6. ¿Qué variable devuelve el número de registros mostrados en cada página?:

- a) PAGE\_NUMBER.
- b) COLUMN\_NUMBER.
- c) REPORT\_COUNT.
- d) PAGE\_COUNT.

7. Para la construcción de una instrucción de filtrado es imprescindible el uso de la palabra:

- a) WHERE.
- b) WHEN.
- c) WHY.
- d) WHAT.

8. Tras la creación de un informe y la conexión de este con las tablas contenedoras de los datos, será posible utilizar estos campos para:

- a) Su importación en un informe.
- b) Su exportación en un informe.
- c) Su exportación en una base de datos.
- d) Su importación en una base de datos.

9. Además de la creación de informes relativos al desarrollo de aplicaciones, para mejorar la legibilidad de la información y garantizar un estructura más óptima podemos realizar también:

- a) Subinformes.
- b) Apartados.
- c) Índices.
- d) Bases de datos.

10. La principal diferencia entre los informes y los subinformes son:

- a) Los subinformes pueden contener más subinformes, no tienen ni encabezados ni pie de página y no tienen existencia por sí solos.
- b) Los informes pueden contener más subinformes, no tienen ni encabezados ni pie de página y no tienen existencia por sí solos.
- c) Los informes no pueden contener más subinformes, no tienen ni encabezados y pie de página y no tienen existencia por sí solos.
- d) Los subinformes no pueden contener más subinformes, no tienen ni encabezados ni pie de página y no tienen existencia por sí solos.

**SOLUCIONES:**

1.  a  b  c  d

2.  a  b  c  d

3.  a  b  c  d

4.  a  b  c  d

5.  a  b  c  d

6.  a  b  c  d

7.  a  b  c  d

8.  a  b  c  d

9.  a  b  c  d

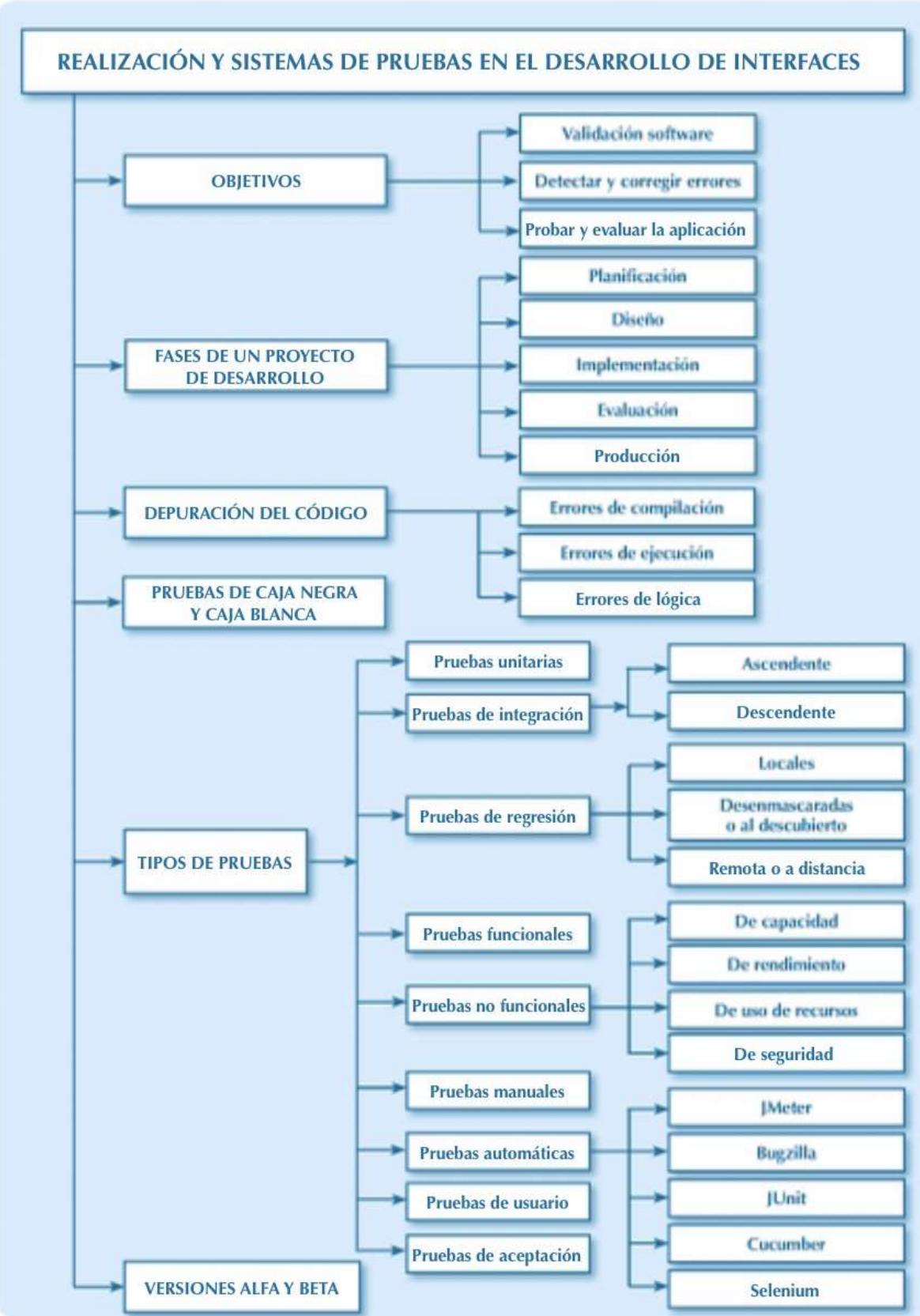
10.  a  b  c  d

# Realización y sistemas de pruebas en el desarrollo de interfaces

## Objetivos

- ✓ Definir una estrategia de pruebas que se adapte al desarrollo particular.
- ✓ Definir y evaluar diferentes tipos de pruebas en las distintas fases del desarrollo.
- ✓ Comprender la importancia de la realización de los distintos tipos de pruebas.
- ✓ Conocer algunas herramientas que se utilizan en la realización de pruebas.
- ✓ Documentar la estrategia de pruebas y los resultados obtenidos.

## Mapa conceptual



## Glosario

**Diseño.** Fase en que comienza la construcción de la aplicación como tal, según las pautas marcadas en la fase de planificación.

**Evaluación.** En la fase de evaluación se comprueba que la funcionalidad del sistema es la adecuada y que cumple el objetivo planteado en un principio. Es una fase muy importante en el desarrollo del proyecto.

**Implementación.** Durante la misma se realiza el desarrollo de la aplicación propiamente dicho, a partir del lenguaje de programación y arquitectura definida en fases anteriores.

**Planificación.** Fase de un proyecto de desarrollo en la que se asientan las bases sobre las que se trabajará durante todo el proceso, con relación al calendario de tiempos, tareas a desarrollar, costes, etc. Será durante esta fase cuando se definan las estrategias de pruebas a realizar.

**Producción.** Fase en la que la aplicación desarrollada se implanta definitivamente en el cliente. Aunque el funcionamiento de la aplicación ya ha sido testeado previamente, se trata de una fase crítica.

**Protocolo de pruebas.** Se trata de una batería de pruebas de diferentes tipos, con distintos fines, que se suelen utilizar como parte del ciclo de desarrollo de una aplicación, y cuyo objetivo final es verificar y asegurar el buen funcionamiento y calidad de una aplicación.

### 10.1. El proyecto de desarrollo

La división en fases de un proyecto va a permitir organizar todo el proceso que conlleva el desarrollo de un nuevo producto, ya sea físico o digital. Es decir, la división en fases aportará un orden lógico a la gestión del proyecto, simplificando la gestión de este en tareas más pequeñas y manejables.

#### 10.1.1. Fases de un proyecto de desarrollo

En cualquier proyecto de desarrollo (también, lógicamente, en el desarrollo de aplicaciones) existe una serie de fases típicas que pueden distinguirse, según se indica en el siguiente gráfico:



**Figura 10.1**  
Fases de un proyecto de desarrollo.

En el desarrollo de estas fases es importante considerar un *protocolo de pruebas* en cada una que ayude a detectar posibles problemas e inconvenientes, antes del paso a la siguiente fase. Es un proceso de especial relevancia en cualquier proyecto, pero especialmente delicado en el ámbito del desarrollo de aplicaciones. La razón es sencilla: detectar un problema de planificación o

diseño en la fase de evaluación o producción puede suponer un aumento de costes considerable, en comparación a su detección en fases más tempranas.

A grandes rasgos, podemos diferenciar las pruebas que se realizan en cada una de las fases o módulos del desarrollo del programa, de las realizadas sobre el funcionamiento general de la aplicación. Es evidente que estas últimas se hacen una vez están finalizados e integrados todos los módulos del sistema, típicamente en la fase de evaluación.

Es muy importante diseñar un protocolo de pruebas exhaustivo, que permita garantizar el correcto funcionamiento de la aplicación y, por tanto, el éxito de esta. A lo largo de este capítulo profundizaremos en muchos tipos de pruebas existentes, analizándolos con mayor detalle.



#### TOMA NOTA

Un correcto diseño y planificación del protocolo de pruebas que se va a seguir es básico para asegurar la funcionalidad y el cumplimiento de los objetivos de un proyecto de desarrollo.

### 10.1.2. Objetivos principales del sistema de pruebas

Como se indicó anteriormente, las fases de pruebas son indispensables para el correcto desarrollo de cualquier aplicación, independientemente de su ámbito. Es fundamental implementar y diseñar un protocolo de pruebas correcto y exhaustivo, en función de cada caso en concreto.

Son múltiples las ventajas que un buen sistema de pruebas aporta al desarrollo de una aplicación:

- Permiten validar el modo en que el software funciona, comparándolo con las especificaciones y objetivos de partida.
- Permiten detectar y corregir errores en el software en fases más tempranas, facilitando así su depuración a lo largo del desarrollo.
- Permite probar y evaluar la aplicación en diferentes escenarios y situaciones.

### 10.1.3. Pruebas de caja negra y caja blanca

Prácticamente la totalidad de los desarrollos de aplicaciones software se basan en dos grupos de pruebas con diferentes características entre sí:

- a) *Pruebas de caja negra*: en las que se evalúa la aplicación desde un punto de vista externo, es decir, sin preocuparnos del “interior”. Son las habituales para la prueba de interfaces.
- b) *Pruebas de caja blanca*: se basan en la evaluación del código interno del software. Un buen diseño para estas pruebas implica la evaluación de todos los posibles caminos que se han implementado en el diseño de un programa.

**Figura 10.2**

Pruebas de caja blanca y pruebas de caja negra.

#### 10.1.4. Depuración de código

Como resultado de realizar pruebas sobre un desarrollo software, será necesario en muchos casos depurar el código, en función de los resultados de dichas pruebas y del objetivo final de la aplicación. El procedimiento habitual suele ser el mostrado en la figura 10.3.

En el caso de la depuración de código, el objetivo principal es detectar y corregir errores en el mismo, en muchos casos a partir de las pruebas realizadas en cada una de las fases del proyecto. Con relación al desarrollo de código, pueden aparecer básicamente tres tipos de errores:

- ✓ *Errores de compilación.* En gran parte de los casos ocurren debido a la sintaxis, que varía en función del lenguaje de programación que se use.
- ✓ *Errores de ejecución.* Suelen aparecer cuando existen operaciones incorrectas, que no son permitidas. Habitualmente, el sistema generará un mensaje de error indicando este motivo.
- ✓ *Errores de lógica.* Están motivados por un error en el diseño del programa, ya que el resultado que se produce no es el esperado. Es más complicado de detectar y, por tanto, de corregir, ya que en este caso el programa se ejecuta de forma normal, sin arrojar ningún error.

**Figura 10.3**

Diagrama de pruebas y depuración de código.

 PARA SABER MÁS

La mayoría de las aplicaciones que se utilizan para el desarrollo de código (como Eclipse o NetBeans) incluyen herramientas de depuración que ayudan a detectar errores.

El proceso de depuración de código va íntimamente ligado a los procesos de pruebas. Según los resultados obtenidos, se intenta localizar el error en el código, tomar en cuenta el mismo para implementar la corrección necesaria (lo que es la depuración de código propiamente dicha) y, por último, volver a probar de nuevo el programa.

### Actividad propuesta 10.1



¿Qué tareas son, a tu juicio, las que se llevan a cabo en las distintas fases de un proyecto de implantación? ¿En qué fase se encuadraría la depuración del código?

## 10.2. Tipos de pruebas

De manera adicional a la clasificación de pruebas como de caja negra o de caja blanca, es posible plantear otros tipos de clasificaciones, según se indica a continuación, según el tipo de funcionalidad que se evalúe en cada caso.

### 10.2.1. Pruebas unitarias

Este primer tipo de pruebas, las *pruebas unitarias*, son las utilizadas para evaluar funcionalidades concretas, examinando todos los caminos posibles implementados en el desarrollo de un algoritmo, función o clase. Por lo tanto, podemos decir que una prueba unitaria es aquella que permite comprobar el funcionamiento de uno de los módulos que forman el programa. Tras evaluar el funcionamiento unitario de la aplicación, se procede con las pruebas en las que se engloban el resto de módulos.

### 10.2.2. Pruebas de integración

Las llamadas *pruebas de integración* se utilizan con el fin de aportar una garantía relacionada con el funcionamiento adecuado de la aplicación, una vez se han integrado todos los módulos que componen la misma, de ahí el nombre de estas pruebas.

Este tipo de pruebas ofrecen la posibilidad de analizar el correcto funcionamiento de todos los módulos de una forma conjunta, ya que es posible que se hayan aplicado otros tipos de pruebas sobre estos módulos, pero de manera separada. Por tanto, la importancia de este tipo de pruebas en el entorno del desarrollo de aplicaciones es fundamental, ya que es habitual que una aplicación de cierta complejidad esté compuesta por varios módulos.

Podemos distinguir dos tipos concretos:

- a) *Pruebas de integración ascendente*. Como su propio nombre indica, este tipo de pruebas consisten en evaluar en primer lugar los niveles o módulos más bajos de la aplicación, para ir subiendo en los mismos de manera gradual.

Estas pruebas se realizan de manera grupal, para lo que es necesario una figura (denominada “controlador”) que se encargue de coordinar dichas pruebas.

- b) *Pruebas de integración descendente.* En este caso, las pruebas se realizarán de manera inversa respecto a las pruebas de integración ascendente, es decir, se comienza desde el módulo principal y se va descendiendo gradualmente.

### 10.2.3. Pruebas de regresión

Las llamadas *pruebas de regresión* se caracterizan por ser un conjunto de pruebas que se habían ejecutado anteriormente sobre el sistema, y que se utilizan para buscar evidencias relacionadas con modificaciones y/o cambios que se haya podido producir sobre el código del programa, con el fin de detectar si han podido ocasionar nuevos errores o fallos, que anteriormente no habían aparecido.

Entre ellas existen diversos tipos, con características diferenciadas entre sí:

- a) *Pruebas de regresión locales:* tienen el fin de encontrar errores que hayan sido ocasionados por cambios o modificaciones recientes en el código.
- b) *Pruebas de regresión desenmascaradas o al descubierto:* en este caso, estas pruebas tienen lugar cuando la realización de cambios ocasiona problemas que no tienen ninguna relación con los mismos, pero que se han detectado a raíz de su inclusión.
- c) *Pruebas de regresión remota o a distancia:* están relacionadas con la aparición de problemas ocasionados al integrar las distintas partes de una aplicación, considerándose que dichas partes por separado no arrojaban ese mismo problema.

Como se puede deducir de este tipo de pruebas, son fundamentales para poder validar de manera correcta todos los cambios que se hagan en el código de la aplicación.



#### Actividad propuesta 10.2

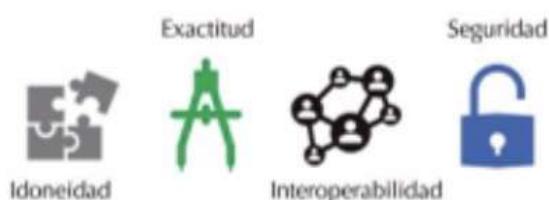
¿Qué tipos de pruebas de regresión conoces? Indica algún ejemplo ilustrativo de las mismas.

### 10.2.4. Pruebas funcionales

Las llamadas *pruebas funcionales* se caracterizan por evaluar al completo las funcionalidades que se indican en la especificación de la aplicación o herramienta desarrollada, concretamente con relación a sus requisitos de diseño.

Al igual que ocurre con el resto de las pruebas que se realicen, es muy importante que el proceso se documente de manera completa, con el fin de facilitar su análisis y comprensión.

Así mismo, la normativa ISO 25010 especifica una serie de características que deben cumplir estas pruebas funcionales para asegurar que estén correctamente definidas. Estas características se recogen en la figura 10.4 y suponen un conjunto de características que sería deseable cumplir.



**Figura 10.4**  
Características deseables de las pruebas funcionales.

Estas pruebas deben ser realizadas por personal con experiencia en su desarrollo para que los resultados puedan ser interpretados de manera adecuada y correcta. Así mismo, es muy importante que se realicen diversas propuestas de mejora si se cree conveniente, o incluso cambios y/o modificaciones si se cree necesario o si los resultados obtenidos no fuesen satisfactorios.

#### Actividad propuesta 10.3



¿Qué características deben cumplir las pruebas funcionales sobre una aplicación? Describelas e indica algún ejemplo.

#### 10.2.5. Pruebas no funcionales: de capacidad, rendimiento, uso de recursos y seguridad

Las *pruebas funcionales* se basan en analizar cómo se comporta la aplicación a nivel interno. A diferencia de estas, las *pruebas no funcionales* suelen utilizarse con el fin de evaluar cómo se comporta la aplicación a nivel externo.

Según lo que hemos estudiado a lo largo de este capítulo, se puede afirmar que las pruebas funcionales son pruebas de caja blanca, mientras que las pruebas no funcionales son pruebas de caja negra.

A continuación, se recogen las principales pruebas no funcionales, junto a sus características más destacadas:

- Prueba de capacidad.* Su uso suele estar ligado a evaluar el comportamiento de la aplicación ante una situación de estrés. Es decir, se utilizan para comprobar la respuesta del sistema ante el aumento de las peticiones o carga de trabajo que recibe, aplicándose una situación de estrés sobre el mismo.
- Prueba de rendimiento.* El fin de este tipo de pruebas es comprobar cómo se comporta la aplicación desde el punto de vista de su eficiencia, teniendo en cuenta parámetros especialmente importantes como el tiempo de respuestas y la velocidad de procesado. Son pruebas muy importantes para decidir si es necesario o no optimizar procesos (y, por tanto, código).

- c) *Prueba de estrés.* Suelen estar relacionadas con las pruebas de capacidad, ya que se basan en lanzar varias peticiones al sistema, con la salvedad de que en este caso lo que se pretende evaluar es cómo se recupera la aplicación ante una situación de sobrecarga, más que la respuesta que se da.
- d) *Prueba de volumen.* También se suelen desarrollar de manera paralela a las anteriores, ya que persiguen evaluar el sistema desde el punto de vista de su capacidad para procesar grandes volúmenes de datos.
- e) *Pruebas de seguridad.* Corresponden a un tipo de pruebas de una gran importancia, ya que tienen como objetivo final aportar una garantía sobre la integridad de los datos de la aplicación. Así mismo, evalúan distintos mecanismos o formas de protección para la aplicación, con el fin de mejorar su robustez y seguridad.

### 10.2.6. Pruebas manuales

Las *pruebas manuales* se caracterizan por ser ejecutadas por parte del encargado de desarrollar el código, con el fin de poder probar su funcionamiento y, en caso de ser necesario, modificar su implementación.

Es decir, para estas pruebas es el mismo desarrollador software el que las ejecuta, evaluando por sí mismo si la respuesta de la aplicación a una determinada entrada es la correcta o no, en función del código implementado con anterioridad.

Para su ejecución no se emplea una herramienta determinada, sino que depende de cada programador. Habitualmente, los entornos de desarrollo más habituales (como Eclipse o NetBeans) incorporan herramientas y/o funcionalidades para facilitar la realización de este tipo de pruebas.

### 10.2.7. Pruebas automáticas

Contrariamente a las pruebas manuales, las *pruebas automáticas* requieren el uso de herramientas de pruebas para poder ejecutarse. En ambos casos, se trata de pruebas complementarias entre sí y que tienen una gran importancia para conseguir un desarrollo software correcto.

Con las pruebas automáticas, su ejecución suele ser más rápida que en el caso de las manuales, ya que, al fin y al cabo, estas últimas las ejecuta una persona. Además, permiten comprobar el funcionamiento de la aplicación ante diversas variaciones en los datos, y repetir las pruebas de una manera rápida y sencilla.

Se suelen utilizar para la realización de pruebas de regresión, ya que consiguen una optimización del procedimiento.

El número y tipo de herramientas existentes en el mercado para llevar a cabo este tipo de pruebas es muy amplio, por lo que el uso de una u otra dependerá del objetivo final, del tipo de prueba, de las preferencias del desarrollador y, por último, de la aplicación que se está desarrollando. A continuación, se destacan algunas de las más utilizadas en el ámbito del desarrollo de aplicaciones software:

- *JMeter:* es una herramienta desarrollada por la prestigiosa compañía Apache, que facilita la realización de pruebas de rendimiento y de carga sobre la aplicación.
- *Bugzilla:* se trata de una aplicación ejecutada online, que se usa para realizar el seguimiento de errores en los módulos del software en cada una de sus versiones.

- *JUnit*: se trata de un grupo de librerías desarrolladas en Java y que se utilizan para realizar pruebas unitarias sobre aplicaciones que han sido desarrolladas en este lenguaje.
- *Cucumber*: es una herramienta de software libre, que facilita realizar pruebas de aceptación sobre aplicaciones web. Se utiliza Ruby como lenguaje para generar scripts.
- *Selenium*: se trata de un grupo de herramientas que se utilizan habitualmente para realizar pruebas de aplicación sobre desarrollos de aplicaciones web.

### 10.2.8. Pruebas de usuario

Las *pruebas de usuario*, tal y como su propio nombre indica, son ejecutadas por uno o varios usuarios reales del sistema o aplicación que se está desarrollando, que no tienen por qué tener conocimientos de programación. Se utilizan de manera complementaria a otros tipos de pruebas que son ejecutadas por expertos, ya que es posible que estos últimos no hayan tenido en cuenta ciertas situaciones que pueden darse en un entorno real de uso de la aplicación. Es decir, son pruebas basadas en la experiencia del usuario en la realización de sus tareas.

Aunque no existe un procedimiento fijado, puede resultar óptimo que participen en este tipo de pruebas al menos quince usuarios, ya que es lógico pensar que cuantos más usuarios prueben la aplicación, más situaciones de error podrán detectarse, aunque tampoco es viable que todos los posibles usuarios de la aplicación participen en las pruebas. En cualquier caso, las pruebas se ejecutan según un guion previo, lo que supone una diferencia sustancial respecto a las pruebas alfa y beta. También puede resultar aconsejable que participen en las mismas distintos tipos de perfiles, para probar a su vez diferentes funcionalidades y procedimientos.

#### RECUERDA

- ✓ Es recomendable que un nuevo desarrollo sea testeado por el mayor número de usuarios posible, pero también es importante que estos usuarios tengan distintos perfiles dentro de la organización. Así se conseguirá maximizar el número de situaciones reales a las que se puede enfrentar la aplicación desarrollada.

### 10.2.9. Pruebas de aceptación

Las llamadas *pruebas de aceptación* se ejecutan sobre una aplicación determinada para corroborar que su funcionamiento cumple con los requerimientos iniciales de diseño, es decir, su funcionamiento es el esperado en el momento del diseño de la aplicación. Es evidente que se trata de un tipo de pruebas de especial relevancia para el éxito de la aplicación.

Estas comprobaciones se realizan desde el prisma del rendimiento de la aplicación, así como desde su funcionalidad, dado que en ambos casos se plantean requerimientos en la fase de diseño del desarrollo. Evidentemente, la aplicación resultante debe satisfacer las necesidades planteadas en cuanto a funcionamiento, pero también ante tiempo de respuesta, comportamiento, estabilidad, etc., con el fin de que la experiencia de uso sea grata y adecuada.

Algunas otras características de este tipo de pruebas son las siguientes:

- ✓ Es habitual que estas pruebas sean definidas por el cliente para el que se está desarrollando la aplicación. Es evidente que el grado de satisfacción sobre el producto dependerá de las expectativas existentes sobre el mismo.
- ✓ Suelen ejecutarse de manera previa a la implantación definitiva de la aplicación.
- ✓ Al igual que ocurre con otros tipos de pruebas, es muy importante que se documenten de manera correcta, recogiendo todas las evidencias posibles sobre las mismas.

### 10.2.10. Desarrollo del plan de pruebas

Como es evidente, el procedimiento de pruebas de cualquier desarrollo software está compuesto por pruebas muy diversas, con características distintas entre sí. Como hemos estudiado, la aplicación de toda esta batería de pruebas es fundamental para garantizar el correcto funcionamiento de la aplicación y, por tanto, tener más posibilidades de éxito.



#### Actividad propuesta 10.4

Según tu criterio, ¿por qué es importante que se siga el desarrollo de pruebas indicado en la figura anterior?

Habitualmente, el flujo que se sigue para ejecutar estas pruebas es el que se muestra en la figura 10.5, aunque este hecho dependerá de cada caso concreto:



**Figura 10.5**  
Desarrollo habitual  
del plan de pruebas.

### 10.3. Versiones alfa y beta

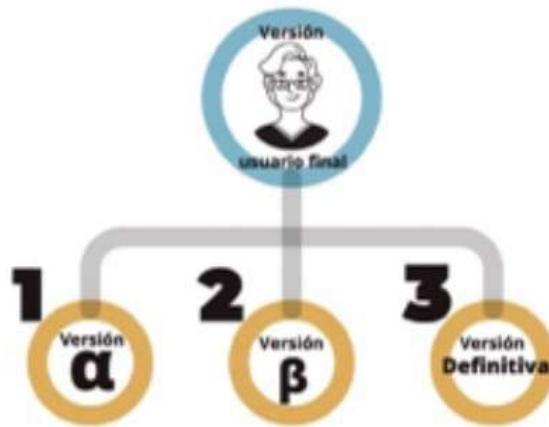
Estas pruebas se utilizan para detectar errores que solo pueden ser descubiertos por parte del usuario final, y que no pueden ser detectados por todas las pruebas realizadas de manera previa

a la puesta en producción de la aplicación. Los tipos de pruebas estudiados hasta ahora en el tema se basan en la mirada del programador, del experto o también del usuario, pero según unas pautas o pasos pre establecidos.

Por tanto, es muy habitual que se creen versiones de pruebas de la aplicación resultante, que son las denominadas versiones alfa y beta, cuyo fin es recoger posibles errores que no hayan sido detectados anteriormente, y que solo pueden recogerse cuando se testeá la aplicación por usuarios reales, en los diferentes entornos y condiciones en los que se vaya a utilizar la misma.

Se trata de conceptos que se suelen utilizar también en otros campos distintos del desarrollo software, como en la fabricación de otros productos o bienes.

En la figura 10.6 se resumen las diferentes versiones que suele tener una aplicación:



**Figura 10.6**  
Tipos de versiones de pruebas.

### 10.3.1. Versión ALFA

Se conoce como *versión alfa* de una aplicación a la primera versión de la misma, que será probada inicialmente por un grupo de personas, cuyo objetivo es simular el uso que le daría el cliente o usuario final. Evidentemente, se trata de una versión que aún no está del todo finalizada para poder implantarse en producción, sino que debe ser previamente depurada y evaluada.

Habitualmente se ejecutan en las mismas oficinas del cliente final para reproducir más fielmente el escenario real de uso.

### 10.3.2. Versión BETA

Se trata de una versión prácticamente final de la aplicación. De hecho, se trata de la primera versión del desarrollo que será probada directamente por el cliente o usuarios finales de la misma, hecho que en sí mismo es la principal diferencia respecto a las pruebas alfa.

Es muy común que se realicen en un entorno de aplicación lo más parecido al real para evitar que el mismo condicione el resultado de su uso.

Los usuarios que prueban estas versiones se conocen como “beta tester”.

#### Actividad propuesta 10.5



- ¿Has utilizado alguna vez alguna versión beta de alguna aplicación?
- ¿Conoces algún ejemplo de uso de estas?

## Resumen

- Como hemos podido estudiar a lo largo del capítulo, la realización y ejecución de pruebas es una fase fundamental, de una gran importancia en el desarrollo de aplicaciones software, sean del tipo que sean. Es evidente que su utilización y ejecución facilitarán la subsanación de errores y fallos del sistema, consiguiendo así un producto mucho más robusto, eficaz y agradable para el usuario.
- Existen multitud de pruebas, según hemos estudiado, aunque se pueden distinguir dos grupos con características distintas entre sí:
  - *Pruebas de caja negra*, que evalúan la aplicación desde un punto de vista externo.
  - *Pruebas de caja blanca*, que evalúan la aplicación desde un punto de vista interno.
- Además de la clasificación anterior, se define otro tipo de pruebas según la función que se pretende evaluar en cada una de ellas. Estas son las pruebas de integración, de regresión, de seguridad, volumen y carga, etc.
- Así mismo, el número de pruebas que se va a realizar no tiene una dependencia directa con el número de líneas de código que tenga la aplicación definitiva, o bien con el número de variables que hayan sido utilizadas en el procedimiento de desarrollo.
- Lo que sí resulta evidente es que, cuanto más tiempo se demore la detección y solución de posibles errores o fallos en el código (es decir, cuanto más tiempo se tarde en la depuración de este), más costará solucionarlo, tanto desde el punto de vista económico como desde el punto de vista del tiempo que será necesario invertir. Además, es evidente que la mayor parte de los fallos y errores que se pueden dar en el desarrollo de una aplicación aparecerán en fases iniciales del mismo, por lo que es muy importante incidir en la realización de pruebas en estas etapas, sin descuidar etapas posteriores.
- Finalmente, también se han analizado los conceptos relacionados con las versiones alfa y beta de una aplicación, que son aquellas que permiten detectar y subsanar determinados errores que únicamente pueden ser detectados por parte del cliente o usuario final.

## Ejercicios propuestos



1. Es completamente imprescindible para abordar cualquier proyecto el realizar una secuencia de fases bien delimitadas que tengan como resultado un producto exitoso o el menos adecuado a las especificaciones del cliente.  
En este ejercicio se pide que se desarrolle un manual de pasos en el que describas la estrategia de diseño de la aplicación, analizando los diferentes pasos o etapas que lo componen.

2. Tras describir la secuencia de fases necesarias para el desarrollo de una aplicación, es necesario centrarse en la fase de pruebas o de evaluación de la aplicación.

Se pide diseñar un plan de pruebas en el que indiques de forma estimada la carga aproximada de tiempo que se van a asignar a cada una de las mismas en un desarrollo de una aplicación software. Esto es importante, puesto que siempre se va a trabajar con plazos de entrega, que han de estar convenientemente estimados y acotados.

3. Se propone realizar un listado de todas las pruebas que se van a realizar para evaluar una aplicación web que hemos desarrollado recientemente, según el encargo realizado por una compañía de venta online de material de oficina, con el fin de poder vender sus productos también de manera online.

Lo aconsejable en este caso sería realizar un listado de todas las pruebas que se van a llevar a cabo para evaluar la aplicación (unitarias, de integración, de sistema). Posteriormente, se elaborará un calendario de tiempos de ejecución, que se adapte a las características de cada prueba.

## ACTIVIDADES DE AUTOEVALUACIÓN

1. Las pruebas de caja blanca:

- a) Se basan en la evaluación del código interno del software, un buen diseño para estas pruebas implica la evaluación de todos los posibles caminos que se han implementado en el diseño de un programa.
- b) Evalúan la aplicación desde un punto de vista externo, es decir, sin preocuparnos del “interior”, son las habituales para la prueba de interfaces.
- c) No permiten validar el funcionamiento del software a partir de las especificaciones de diseño.
- d) Ninguna de las respuestas anteriores es correcta.

2. ¿Cuál de los siguientes tipos de pruebas de software no es habitual?:

- a) Pruebas unitarias.
- b) Pruebas de integración.
- c) Pruebas de inmersión.
- d) Pruebas de seguridad.

3. ¿Qué prueba se centra sobre todo en la evaluación de los sistemas de protección y autenticación de una aplicación?:

- a) Pruebas de seguridad.
- b) Pruebas de integración.
- c) Pruebas de volumen y carga.
- d) Pruebas unitarias.

4. La depuración de código sirve para:

- a) La revisión del código para la detección de posibles errores y la corrección de los mismos.
- b) Crear errores de sintaxis.
- c) Producir fallos de diseño.
- d) Interpretar las instrucciones correctas.

5. Las pruebas de integración ascendente:

- a) Suele tener una integración primero en profundidad y otra integración primero en anchura.
- b) Se realiza desde el módulo principal hasta los subordinados.
- c) Comienzan con la evaluación de los niveles más altos.
- d) Comienzan con la evaluación de los niveles más bajos.

6. Las pruebas de regresión:

- a) Son los errores producidos cuando al realizar la integración de las diferentes partes de un programa se producen errores que de forma individual no ocurrían.
- b) Solo se producen cuando la introducción de cambios muestra errores que nada tienen que ver con las modificaciones realizadas.
- c) Solo localizan errores que se han producido por la introducción de cambios.
- d) Es la nueva ejecución de un conjunto de pruebas que ya había sido ejecutado con anterioridad.

7. Las pruebas de capacidad:

- a) Se utilizan para la evaluación de la capacidad de procesamiento del software ante la llegada de una cantidad grande de datos.
- b) Se utilizan para la evaluación de la capacidad de recuperación del software ante una sobrecarga de datos.
- c) Se utilizan para la evaluación del tiempo de respuesta y la velocidad de procesamiento del software.
- d) Se utilizan para la evaluación del software y su comportamiento ante un aumento de peticiones, es decir, ante un incremento de la carga de trabajo.

8. Las pruebas de rendimiento:

- a) Se utilizan para la evaluación de la capacidad de procesamiento del software ante la llegada de una cantidad grande de datos.
- b) Se utilizan para la evaluación de la capacidad de recuperación del software ante una sobrecarga de datos.
- c) Se utilizan para la evaluación del tiempo de respuesta y la velocidad de procesamiento del software.
- d) Se utilizan para la evaluación del software y su comportamiento ante un aumento de peticiones, es decir, ante un incremento de la carga de trabajo.

9. Los test de usuario se basan en pruebas que observan la forma de interacción de los usuarios con el producto objeto del test según el:

- a) Diseño directo en la herramienta (DDH).
- b) Diseño centrado en la yuxtaposición (DC).
- c) Diseño centrado en el usuario (DCU).
- d) Diseño centrado en el experto (DCE).

10. La versión alfa de un producto:

- a) Consiste en la primera versión de la aplicación.
- b) Consiste en la última versión de la aplicación.
- c) Consiste en la versión casi definitiva de la aplicación.
- d) Se prueba por los usuarios de tipo beta tester.

**SOLUCIONES:**

1. a b c d

2. a b c d

3. a b c d

4. a b c d

5. a b c d

6. a b c d

7. a b c d

8. a b c d

9. a b c d

10. a b c d