**Libro: “Aprenda Java como si estuviera en primero”.**

**Conceptos Generales:**

**Introducción a JAVA:**

Java surge en 1991 cuando un grupo de ingenieros de Sun Microsystems trataron de diseñar un nuevo lenguaje de programación destinado a electrodomésticos. La reducida potencia de cálculo y memoria de los electrodomésticos llevó a desarrollar un lenguaje sencillo capaz de generar código de tamaño muy reducido.

Java, como lenguaje de programación para computadores, se introdujo a finales de 1995. La clave fue la incorporación de un intérprete Java en el programa Netscape Navigator, versión 2.0, produciendo una verdadera revolución en Internet. Java 1.1 apareció a principios de 1997, mejorando sustancialmente la primera versión del lenguaje.

Java incorpora muchos aspectos que en cualquier otro lenguaje son extensiones propiedad de empresas de software o fabricantes de ordenadores (threads, ejecución remota, componentes, seguridad, acceso a bases de datos, etc.). Por eso es un lenguaje ideal para aprender la informática moderna, porque incorpora todos estos conceptos de un modo estándar, mucho más sencillo y claro que con las citadas extensiones de otros lenguajes. Esto es consecuencia de haber sido diseñado más recientemente y por un único equipo.

El principal objetivo del lenguaje Java es llegar a ser el “nexo universal” que conecte a los usuarios con la información, esté ésta situada en el ordenador local, en un servidor de Web, en una base de datos o en cualquier otro lugar.

**El entorno de desarrollo de JAVA:**

La compañía Sun, creadora de Java, distribuye gratuitamente el Java(tm) Development Kit (JDK). Se trata de un conjunto de programas y librerías que permiten desarrollar, compilar y ejecutar programas en Java. Incorpora además la posibilidad de ejecutar parcialmente el programa, deteniendo la ejecución en el punto deseado y estudiando en cada momento el valor de cada una de las variables (es el denominado Debugger). Existe también una versión reducida del JDK, denominada JRE (Java Runtime Environment) destinada únicamente a ejecutar código Java (no permite compilar).

Los IDEs (Integrated Development Environment), tal y como su nombre indica, son entornos de desarrollo integrados. En un mismo programa es posible escribir el código Java, compilarlo y ejecutarlo sin tener que cambiar de aplicación. Algunos incluyen una herramienta para realizar Debug gráficamente, frente a la versión que incorpora el JDK basada en la utilización de una consola (denominada habitualmente ventana de comandos de MS-DOS, en Windows NT/95/98) bastante difícil y pesada de utilizar. Estos entornos integrados permiten desarrollar las aplicaciones de forma mucho más rápida, incorporando en muchos casos librerías con componentes ya desarrollados, los cuales se incorporan al proyecto o programa. Como inconvenientes se pueden señalar algunos fallos de compatibilidad entre plataformas y ficheros resultantes de mayor tamaño que los basados en clases estándar.

**El compilador de JAVA:**

Se trata de una de las herramientas de desarrollo incluidas en el JDK. Realiza un análisis de sintaxis del código escrito en los ficheros fuente de Java (con extensión \*.java). Si no encuentra errores en el código genera los ficheros compilados (con extensión \*.class). En otro caso muestra la línea o líneas erróneas. En el JDK de Sun dicho compilador se llama javac.exe. Tiene numerosas opciones, algunas de las cuales varían de una versión a otra. Se aconseja consultar la documentación de la versión del JDK utilizada para obtener una información detallada de las distintas posibilidades.

**La Java Virtual Machine (JVM):**

Se plantea la necesidad de conseguir un código capaz de ejecutarse en cualquier tipo de máquina. Una vez compilado no debería ser necesaria ninguna modificación por el hecho de cambiar de procesador o de ejecutarlo en otra máquina. La clave consistió en desarrollar un código “neutro” el cual estuviera preparado para ser ejecutado sobre una “máquina hipotética o virtual”, denominada Java Virtual Machine (JVM). Es esta JVM quien interpreta este código neutro convirtiéndolo a código particular de la CPU o chip utilizada. Se evita tener que realizar un programa diferente para cada CPU o plataforma. La JVM es el intérprete de Java. Ejecuta los “bytecodes” (ficheros compilados con extensión \*.class) creados por el compilador de Java (javac.exe). Tiene numerosas opciones entre las que destaca la posibilidad de utilizar el denominado JIT (Just-In-Time Compiler), que puede mejorar entre 10 y 20 veces la velocidad de ejecución de un programa.

**Nomenclatura habitual en la programación en JAVA:**

Los nombres de Java son sensibles a las letras mayúsculas y minúsculas. Así, las variables masa, Masa y MASA son consideradas variables completamente diferentes.

**Convenciones:**

1. En Java es habitual utilizar nombres con minúsculas, con las excepciones que se indican en los puntos siguientes.

2. Cuando un nombre consta de varias palabras es habitual poner una a continuación de otra, poniendo con mayúscula la primera letra de la palabra que sigue a otra (Ejemplos: elMayor(), VentanaCerrable, RectanguloGrafico, addWindowListener()).

3. Los nombres de clases e interfaces comienzan siempre por mayúscula (Ejemplos: Geometria, Rectangulo, Dibujable, Graphics, Vector, Enumeration).

4. Los nombres de objetos, los nombres de métodos y variables miembro, y los nombres de las variables locales de los métodos, comienzan siempre por minúscula (Ejemplos: main(), dibujar(), numRectangulos, x, y, r).

5. Los nombres de las variables finales, es decir de las constantes, se definen siempre con mayúsculas (Ejemplo: PI)

**Estructura general de un programa JAVA:**

Los ficheros fuente tienen la extensión \*.java, mientras que los ficheros compilados tienen la extensión \*.class.

Un fichero fuente (\*.java) puede contener más de una clase, pero sólo una puede ser public. El nombre del fichero fuente debe coincidir con el de la clase public (con la extensión \*.java). Si por ejemplo en un fichero aparece la declaración (public class MiClase {...}) entonces el nombre del fichero deberá ser MiClase.java. Es importante que coincidan mayúsculas y minúsculas ya que MiClase.java y miclase.java serían clases diferentes para Java. Si la clase no es public, no es necesario que su nombre coincida con el del fichero. Una clase puede ser public o package (default), pero no private o protected. Estos conceptos se explican posteriormente.

De ordinario una aplicación está constituida por varios ficheros \*.class. Cada clase realiza unas funciones particulares, permitiendo construir las aplicaciones con gran modularidad e independencia entre clases. La aplicación se ejecuta por medio del nombre de la clase que contiene la función main() (sin la extensión \*.class). Las clases de Java se agrupan en packages, que son librerías de clases. Si las clases no se definen como pertenecientes a un package, se utiliza un package por defecto (default) que es el directorio activo.

**Concepto de Clase:**

Una clase es una agrupación de datos (variables o campos) y de funciones (métodos) que operan sobre esos datos. A estos datos y funciones pertenecientes a una clase se les denomina variables y métodos o funciones miembro. La programación orientada a objetos se basa en la programación de clases. Un programa se construye a partir de un conjunto de clases.

Una vez definida e implementada una clase, es posible declarar elementos de esta clase de modo similar a como se declaran las variables del lenguaje (int, double, String, …). Los elementos declarados de una clase se denominan objetos de la clase. De una única clase se pueden declarar o crear numerosos objetos. La clase es lo genérico: es el patrón o modelo para crear objetos. Cada objeto tiene sus propias copias de las variables miembro, con sus propios valores, en general distintos de los demás objetos de la clase. Las clases pueden tener variables static, que son propias de la clase y no de cada objeto.

**Herencia:**

La herencia permite que se pueden definir nuevas clases basadas en clases existentes, lo cual facilita re-utilizar código previamente desarrollado. Si una clase deriva de otra (extends) hereda todas sus variables y métodos. La clase derivada puede añadir nuevas variables y métodos y/o redefinir las variables y métodos heredados. En Java, a diferencia de otros lenguajes orientados a objetos, una clase sólo puede derivar de una única clase, con lo cual no es posible realizar herencia múltiple en base a clases. Sin embargo, es posible “simular” la herencia múltiple en base a las interfaces.

**Concepto de Interface:**

Una interface es un conjunto de declaraciones de funciones. Si una clase implementa (implements) una interface, debe definir todas las funciones especificadas por la interface. Una clase puede implementar más de una interface, representando una forma alternativa de la herencia múltiple. Una interface puede derivar de otra o incluso de varias interfaces, en cuyo caso incorpora todos los métodos de las interfaces de las que deriva.

**Concepto de Package:**

Un package es una agrupación de clases. Existen una serie de packages incluidos en el lenguaje (ver jerarquía de clases que aparece en el API de Java). Además el usuario puede crear sus propios packages. Lo habitual es juntar en packages las clases que estén relacionadas. Todas las clases que formen parte de un package deben estar en el mismo directorio.

**La jerarquía de clases de Java (API)**

Durante la generación de código en Java, es recomendable y casi necesario tener siempre a la vista la documentación on-line del API de Java 1.1 ó Java 1.2. En dicha documentación es posible ver tanto la jerarquía de clases, es decir la relación de herencia entre clases, como la información de los distintos packages que componen las librerías base de Java.

Es importante distinguir entre lo que significa herencia y package. Un package es una agrupación arbitraria de clases, una forma de organizar las clases. La herencia sin embargo consiste en crear nuevas clases en base a otras ya existentes. Las clases incluidas en un package no derivan en general de la misma clase.

En la documentación on-line se presentan ambas visiones: “Package Index” y “Class Hierarchy”. La primera presenta la estructura del API de Java agrupada por packages, mientras que en la segunda aparece la jerarquía de clases. Hay que resaltar el hecho de que todas las clases en Java son derivadas de la clase java.lang.Object, por lo que heredan todos los métodos y variables de ésta.

Si se selecciona una clase en particular, la documentación muestra una descripción detallada de todos los métodos y variables de la clase. A su vez muestra su herencia completa (partiendo de la clase java.lang.Object).

**Variables:**

Una variable es un nombre que contiene un valor que puede cambiar a lo largo del programa. De acuerdo con el tipo de información que contienen, en Java hay dos tipos principales de variables:

1. Variables de tipos primitivos. Están definidas mediante un valor único.

2. Variables referencia. Las variables referencia son referencias o nombres de una información más compleja: arrays u objetos de una determinada clase.

Desde el punto de vista de su papel en el programa, las variables pueden ser:

1. Variables miembro de una clase: Se definen en una clase, fuera de cualquier método; pueden ser tipos primitivos o referencias.

2. Variables locales: Se definen dentro de un método o más en general dentro de cualquier bloque entre llaves {}. Se crean en el interior del bloque y se destruyen al finalizar dicho bloque. Pueden ser también tipos primitivos o referencias.

**Tipos primitivos de variables:**

Se llaman tipos primitivos de variables de Java a aquellas variables sencillas que contienen los tipos de información más habituales: valores boolean, caracteres y valores numéricos enteros o de punto flotante. Java dispone de ocho tipos primitivos de variables: un tipo para almacenar valores true y false (boolean); un tipo para almacenar caracteres (char), y 6 tipos para guardar valores numéricos, cuatro tipos para enteros (byte, short, int y long) y dos para valores reales de punto flotante (float y double). Los rangos y la memoria que ocupa cada uno de estos tipos se muestran en la Tabla 2.1.

Texto

Descripción generada automáticamente

Los tipos primitivos de Java tienen algunas características importantes que se resumen a continuación:

1. El tipo boolean no es un valor numérico: sólo admite los valores true o false. El tipo boolean no se identifica con el igual o distinto de cero, como en C/C++. El resultado de la expresión lógica que aparece como condición en un bucle o en una bifurcación debe ser boolean.

2. El tipo char contiene caracteres en código UNICODE (que incluye el código ASCII), y ocupan 16 bits por carácter. Comprende los caracteres de prácticamente todos los idiomas.

3. Los tipos byte, short, int y long son números enteros que pueden ser positivos o negativos, con distintos valores máximos y mínimos. A diferencia de C/C++, en Java no hay enteros unsigned.

4. Los tipos float y double son valores de punto flotante (números reales) con 6-7 y 15 cifras decimales equivalentes, respectivamente.

5. Se utiliza la palabra void para indicar la ausencia de un tipo de variable determinado.

6. A diferencia de C/C++, los tipos de variables en Java están perfectamente definidos en todas y cada una de las posibles plataformas. Por ejemplo, un int ocupa siempre la misma memoria y tiene el mismo rango de valores, en cualquier tipo de ordenador.

7. Extensiones de Java 1.2 para aprovechar la arquitectura de los procesadores Intel, que permiten realizar operaciones con una precisión extendida de 80 bits

Otra característica del lenguaje es que es posible declarar una variable dentro de un bloque con el mismo nombre que una variable miembro, pero no con el nombre de otra variable local. La variable declarada dentro del bloque oculta a la variable miembro en ese bloque. Para acceder a la variable miembro oculta será preciso utilizar el operador this.

Uno de los aspectos más importantes en la programación orientada a objetos (OOP) es la forma en la cual son creados y eliminados los objetos. La forma de crear nuevos objetos es utilizar el operador new. Cuando se utiliza el operador new, la variable de tipo referencia guarda la posición de memoria donde está almacenado este nuevo objeto. Para cada objeto se lleva cuenta de por cuántas variables de tipo referencia es apuntado. La eliminación de los objetos la realiza el denominado garbage collector, quien automáticamente libera o borra la memoria ocupada por un objeto cuando no existe ninguna referencia apuntando a ese objeto. Lo anterior significa que aunque una variable de tipo referencia deje de existir, el objeto al cual apunta no es eliminado si hay otras referencias apuntando a ese mismo objeto.

**Casos especiales: Clases BigInteger y BigDecimal**

Java 1.1 incorporó dos nuevas clases destinadas a operaciones aritméticas que requieran gran precisión: BigInteger y BigDecimal. La forma de operar con objetos de estas clases difiere de las operaciones con variables primitivas. En este caso hay que realizar las operaciones utilizando métodos propios de estas clases (add() para la suma, subtract() para la resta, divide() para la división, etc.). Se puede consultar el package java.math donde aparecen ambas clases con todos sus métodos.

Los objetos de tipo BigInteger son capaces de almacenar cualquier número entero sin perder información. Esto significa que es posible trabajar con enteros de cualquier número de cifras sin perder información durante las operaciones. Análogamente los objetos de tipo BigDecimal permiten trabajar con el número de decimales deseado.

Operador condicional ?:

Este operador, tomado de C/C++, permite realizar bifurcaciones condicionales sencillas. Su forma general es la siguiente:

booleanExpression ? res1 : res2

donde se evalúa booleanExpression y se devuelve res1 si el resultado es true y res2 si el resultado es false. Es el único operador ternario (tres argumentos) de Java. Como todo operador que devuelve un valor puede ser utilizado en una expresión.

Por ejemplo las sentencias:

x=1; y=10; z = (x<y) ? x+3 : y+8;

asignarían a z el valor 4, es decir x+3.

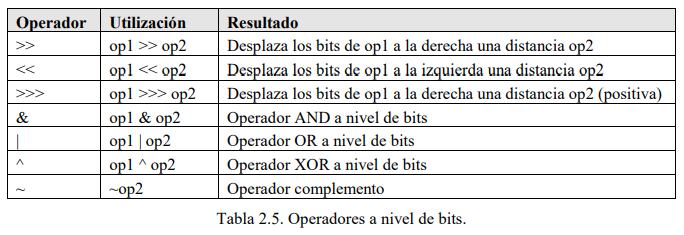
**Operadores lógicos:**

**Tabla

Descripción generada automáticamente**

**Operadores que actúan a nivel de bits**

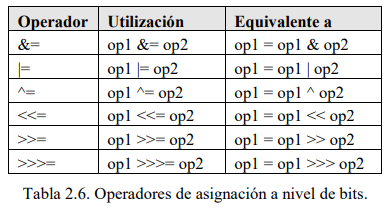
Java dispone también de un conjunto de operadores que actúan a nivel de bits. Las operaciones de bits se utilizan con frecuencia para definir señales o flags, esto es, variables de tipo entero en las que cada uno de sus bits indican si una opción está activada o no. La Tabla 2.5 muestra los operadores de Java que actúan a nivel de bits.



En binario, las potencias de dos se representan con un único bit activado. Por ejemplo, los números (1, 2, 4, 8, 16, 32, 64, 128) se representan respectivamente de modo binario en la forma (00000001, 00000010, 00000100, 00001000, 00010000, 00100000, 01000000, 10000000), utilizando sólo 8 bits. La suma de estos números permite construir una variable flags con los bits activados que se deseen. Por ejemplo, para construir una variable flags que sea 00010010 bastaría hacer flags=2+16. Para saber si el segundo bit por la derecha está o no activado bastaría utilizar la sentencia,

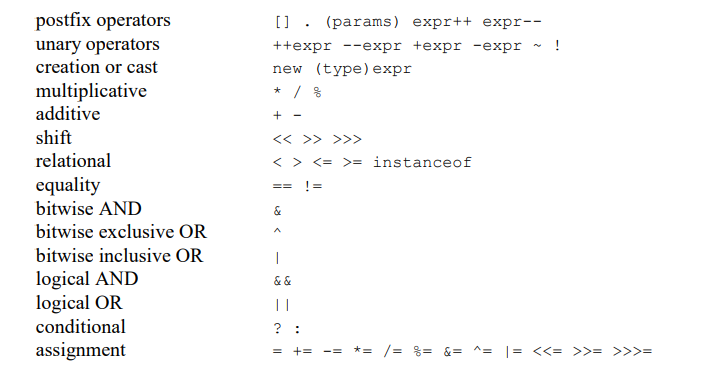
if (flags & 2 == 2) {...}

La Tabla 2.6 muestra los operadores de asignación a nivel de bits.



**Precedencia de operadores**

El orden en que se realizan las operaciones es fundamental para determinar el resultado de una expresión. Por ejemplo, el resultado de x/y\*z depende de qué operación (la división o el producto) se realice primero. La siguiente lista muestra el orden en que se ejecutan los distintos operadores en un sentencia, de mayor a menor precedencia:



En Java, todos los operadores binarios, excepto los operadores de asignación, se evalúan de izquierda a derecha. Los operadores de asignación se evalúan de derecha a izquierda, lo que significa que el valor de la izquierda se copia sobre la variable de la derecha.

**Bifurcaciones:**

Bifurcación if:

Esta estructura permite ejecutar un conjunto de sentencias en función del valor que tenga la expresión de comparación (se ejecuta si la expresión de comparación tiene valor true). Tiene la forma siguiente:

if (booleanExpression) {

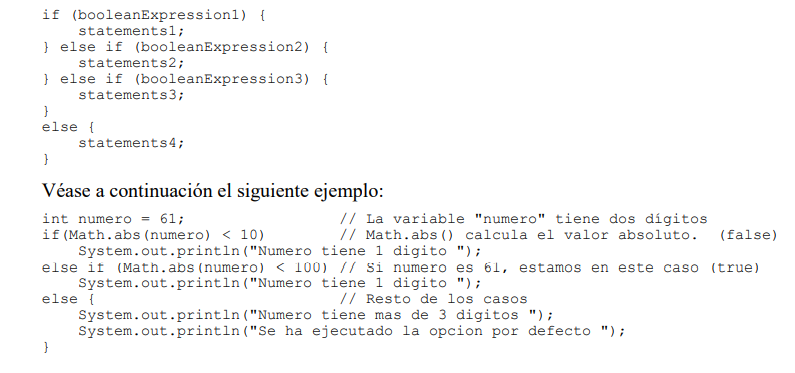
statements;

}

Las llaves {} sirven para agrupar en un bloque las sentencias que se han de ejecutar, y no son necesarias si sólo hay una sentencia dentro del if.

Bifurcación if elseif else:

Permite introducir más de una expresión de comparación. Si la primera condición no se cumple, se compara la segunda y así sucesivamente. En el caso de que no se cumpla ninguna de las comparaciones se ejecutan las sentencias correspondientes al else.



Sentencia switch:

Se trata de una alternativa a la bifurcación if elseif else cuando se compara la misma expresión con distintos valores. Su forma general es la siguiente:

Texto

Descripción generada automáticamente con confianza media

Las características más relevantes de switch son las siguientes:

1. Cada sentencia case se corresponde con un único valor de expression. No se pueden establecer rangos o condiciones sino que se debe comparar con valores concretos. El ejemplo del Apartado 2.3.3.3 no se podría realizar utilizando switch.

2. Los valores no comprendidos en ninguna sentencia case se pueden gestionar en default, que es opcional.

3. En ausencia de break, cuando se ejecuta una sentencia case se ejecutan también todas las que van a continuación, hasta que se llega a un break o hasta que se termina el switch.

Ejemplo:

Interfaz de usuario gráfica, Texto

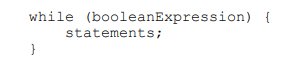
Descripción generada automáticamente

Bucles:

Un bucle se utiliza para realizar un proceso repetidas veces. Se denomina también lazo o loop. El código incluido entre las llaves {} (opcionales si el proceso repetitivo consta de una sola línea), se ejecutará mientras se cumpla unas determinadas condiciones. Hay que prestar especial atención a los bucles infinitos, hecho que ocurre cuando la condición de finalizar el bucle (booleanExpression) no se llega a cumplir nunca. Se trata de un fallo muy típico, habitual sobre todo entre programadores poco experimentados.

Bucle while

Las sentencias statements se ejecutan mientras booleanExpression sea true.



Bucle for

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

La sentencia o sentencias initialization se ejecuta al comienzo del for, e increment después de statements. La booleanExpression se evalúa al comienzo de cada iteración; el bucle termina cuando la expresión de comparación toma el valor false. Cualquiera de las tres partes puede estar vacía. La initialization y el increment pueden tener varias expresiones separadas por comas. Por ejemplo, el código situado a la izquierda produce la salida que aparece a la derecha:

Gráfico

Descripción generada automáticamente

Bucle do while

Es similar al bucle while pero con la particularidad de que el control está al final del bucle (lo que hace que el bucle se ejecute al menos una vez, independientemente de que la condición se cumpla o no). Una vez ejecutados los statements, se evalúa la condición: si resulta true se vuelven a ejecutar las sentencias incluidas en el bucle, mientras que si la condición se evalúa a false finaliza el bucle.

Texto

Descripción generada automáticamente

La sentencia break es válida tanto para las bifurcaciones como para los bucles. Hace que se salga inmediatamente del bucle o bloque que se está ejecutando sin finalizar el resto de las sentencias. La sentencia continue se utiliza en los bucles (no en bifurcaciones). Finaliza la iteración “i” que en ese momento se está ejecutando (no ejecuta el resto de sentencias que hubiera hasta el final del bucle). Vuelve al comienzo del bucle y comienza la siguiente iteración (i+1).

Sentencias break y continue con etiquetas:

Las etiquetas permiten indicar un lugar donde continuar la ejecución de un programa después de un break o continue. El único lugar donde se pueden incluir etiquetas es justo delante de un bloque de código entre llaves {} (if, switch, do...while, while, for) y sólo se deben utilizar cuando se tiene uno o más bucles (o bloques) dentro de otro bucle y se desea salir (break) o continuar con la siguiente iteración (continue) de un bucle que no es el actual.

Interfaz de usuario gráfica, Texto, Aplicación

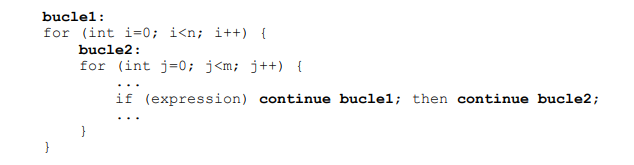
Descripción generada automáticamente

la expresión break bucleI; finaliza los dos bucles simultáneamente, mientras que la expresión break; sale del bucle while interior y seguiría con el bucle for en i. Con los valores presentados ambos bucles finalizarán con i = 5 y j = 6 (se invita al lector a comprobarlo).

La sentencia continue (siempre dentro de al menos un bucle) permite transferir el control a un bucle con nombre o etiqueta. Por ejemplo, la sentencia,

Continue bucle1;

transfiere el control al bucle for que comienza después de la etiqueta bucle1: para que realice una nueva iteración:



Java Code Conventions:

Java Source Files

Each Java source file contains a single public class or interface. When private classes and interfaces are associated with a public class, you can put them in the same source file as the public class. The public class should be the first class or interface in the file.

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

COMMENTS