PROGRAMACIÓN UD-9

INTERFACES

ÍNDICE

[INTRODUCCIÓN 3](#_Toc127432229)

[CONCEPTO DE INTERFAZ 3](#_Toc127432230)

[A. DEFINICIÓN DE UNA INTERFAZ 3](#_Toc127432231)

[B. IMPLEMENTACIÓN DE UNA INTERFAZ 3](#_Toc127432232)

[ATRIBUTOS DE UNA INTERFAZ 4](#_Toc127432233)

[MÉTODOS IMPLEMENTADOS EN UNA INTERFAZ 5](#_Toc127432234)

[A. MÉTODOS POR DEFECTO 5](#_Toc127432235)

[B. MÉTODOS ESTÁTICOS EN UNA INTERFAZ 6](#_Toc127432236)

[C. MÉTODOS PRIVADOS 7](#_Toc127432237)

[HERENCIA 7](#_Toc127432238)

[VARIABLES DE TIPO INTERFAZ 7](#_Toc127432239)

[CLASES ANÓNIMAS 9](#_Toc127432240)

[ACCESO ENTRE MIEMBROS DE UNA INTERFAZ 9](#_Toc127432241)

[SINTAXIS GENERAL 9](#_Toc127432242)

[UN PAR DE INTERFACES DE LA API 11](#_Toc127432243)

[A. INTERFAZ COMPARABLE 11](#_Toc127432244)

[B. INTERFAZ COMPARATOR 14](#_Toc127432245)

# INTRODUCCIÓN

Supongamos que vamos a trabajar con clases de animales. De ellos, unos tienen la capacidad de emitir un sonido, por ejemplo, los perros, los gatos o los lobos, y otros no. Los primeros, por tanto, tendrán el método void voz().

Sin embargo, la forma en que se ejecuta dicho método es distinta según la clase. En un objeto de la clase Gato, la ejecución de voz () hará que aparezca en la pantalla la cadena ¡Miau! En cambio, en un Perro ¡Guau! Pero todos ellos tienen en común que implementan el método voz().

# CONCEPTO DE INTERFAZ

Para gestionar la situación descrita en la introducción, podríamos crear la clase abstracta Animal, de la que heredarían las clases Gato y Perro, con el método abstracto voz(), pero eso descartaría otras clases que no son animales pero emiten sonidos como Persona o Piano.

En Java disponemos de otra forma de expresar una funcionalidad común de algunas clases que no tienen relación entre ellas, en este caso las que tienen implementado el método voz().

A dichas funcionalidades las definiremos en las **interfaces**. En nuestro ejemplo hablaríamos de la interfaz **Sonido**, que consiste en implementar el método voz(). Diremos que las clases Perro, Gato y Lobo implementan la interfaz Sonido, ya que, entre sus métodos está voz(), mientras que las clases Caracol y Lagartija no.

## DEFINICIÓN DE UNA INTERFAZ

La definición de la interfaz Sonido tendría la siguiente forma:

Imagen que contiene Texto

Descripción generada automáticamente

En la definición aparece el nombre del método, la lista de parámetros de entrada aquí está vacía, y el tipo devuelto void. Naturalmente, no se implementa el cuerpo del método, ya que este depende de la clase concreta que implemente Sonido.

## IMPLEMENTACIÓN DE UNA INTERFAZ

Es en la definición de cada clase donde se decide qué hace exactamente voz(). Por ejemplo, la clase Perro se definiría así:

Texto

Descripción generada automáticamente

Como vemos, cuando una clase implementa una interfaz, la declara en la cabecera con la palabra reservada **implements** seguida del nombre de la interfaz, en nuestro caso, Sonido. Con ello estamos declarando que la clase Perro implementa la interfaz Sonido y que, por tanto, entre sus métodos se encuentra voz(). La implementación de un método de una interfaz en una clase tiene que declararse public, como hemos hecho en nuestro ejemplo. La definición de la clase Gato sería:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

En cambio, la clase Caracol no implementa la interfaz Sonido y entre sus métodos no encontraremos la implementación de voz().

En definitiva, cuando una clase declara en su encabezamiento que tiene implementada una interfaz concreta, se está comprometiendo a tener implementadas en su definición todas las funcionalidades de esa interfaz. En caso contrario, el compilador se encargará de mostrar un mensaje de error. Las interfaces, sin embargo, no son instanciables por sí mismas; sólo podemos crear objetos de las clases que las implementan.

# ATRIBUTOS DE UNA INTERFAZ

Una interfaz también puede declarar atributos. Por ejemplo, si queremos llevar la cuenta de las sucesivas versiones de nuestra interfaz Sonido podemos definir el atributo entero **version**. La definición de la interfaz tendría la siguiente forma:

Texto

Descripción generada automáticamente

El atributo version se convertirá automáticamente en un atributo de las clases Gato y Perro, y solo podrá cambiarse en la definición de Sonido, pero no en las clases que la implementan, ya que los atributos definidos en una interfaz son static y final por defecto, sin necesidad de escribirlo explícitamente

Se accede a ellos, solo para lectura, a través del nombre de la interfaz o de una clase que la implemente, pero no desde una instancia. Por ejemplo, si queremos mostrar la versión actual de la interfaz Sonido, podemos poner



O bien



Una interfaz puede incluso constar solo de atributos. Se pueden usar para añadir un conjunto de constantes compartidas por muchas clases distintas.

# MÉTODOS IMPLEMENTADOS EN UNA INTERFAZ

Hasta ahora solo hemos declarado un método en la interfaz Sonido. Pero, en general, una interfaz puede declarar más de uno. Entre ellos puede a ver también métodos implementados en la propia interfaz. A los métodos declarados, pero no implementados, como voz(), se les llama abstractos, igual que en las clases abstractas. Los que sí están implementados en la interfaz se llaman métodos de extensión y pueden ser tanto estáticos como no estáticos, públicos o privados.

A los métodos de extensión no estáticos públicos se les llama métodos por defecto.

## MÉTODOS POR DEFECTO

Los métodos por defecto se declaran anteponiendo la palabra reservada default y son public sin necesidad de especificarlo Vamos a añadir uno a nuestro ejemplo de los animales. Supongamos que, entre los sonidos de los animales, queremos incluir los ruidos que hacen durmiendo, y que todos emiten el mismo sonido cuando duermen. En este caso podemos incluir, dentro de la interfaz, el método vozDurmiendo() que va a valer para todos los animales. Por tanto, lo implementamos en la misma interfaz.

Texto

Descripción generada automáticamente

Este método, con su implementación, será incorporado por Perro y Gato automáticamente, y será accesible para todos los objetos de ambas clases.

Texto

Descripción generada automáticamente

No obstante, vozDurmiendo() se puede reimplementar en cada clase haciendo overriding de la implementación que se ha hecho en la interfaz Supongamos que los leones son la excepción entre los animales, y rugen hasta cuando duermen.

Texto

Descripción generada automáticamente

En este caso, el código

Texto

Descripción generada automáticamente

Mostraría por consola ¡Grrrr!

## MÉTODOS ESTÁTICOS EN UNA INTERFAZ

En una interfaz también se pueden implementar métodos estáticos. Si no se especifica un modificador de acceso, son public. Estos métodos pertenecen a la interfaz, y no a las clases que la implementan y mucho menos a sus objetos. Por ejemplo, supongamos que todos los animales, sin excepción, emitieran el mismo sonido al bostezar. Entonces podríamos implementar un método estático para los bostezos.

Texto

Descripción generada automáticamente

Este método será accesible directamente desde la interfaz Sonido. Para invocarlo tendremos que escribir

Imagen que contiene Logotipo

Descripción generada automáticamente

## MÉTODOS PRIVADOS

Además de los métodos abstractos, por defecto y estáticos, que son públicos por defecto, en una interfaz se pueden implementar métodos privados anteponiendo el modificador private. Pueden ser tanto estáticos como no estáticos y no son accesibles fuera del código de la interfaz.

Estos métodos están implementados, no son abstractos, y solo pueden ser invocados por el resto de los métodos no abstractos de la interfaz. En general, son métodos auxiliares para uso interno de otros métodos de la interfaz.

# HERENCIA

Las interfaces pueden heredar unas de otras, heredando todos los atributos y métodos, salvo los privados, y pudiéndose añadir otros nuevos. A diferencia de las clases, una interfaz puede heredar de más de una interfaz, entre las interfaces es posible la herencia múltiple.

Veamos un ejemplo de interfaz que hereda de otra.

A partir de la interfaz Cola podemos definir la interfaz ColaDoble que añade dos métodos nuevos, que encolan en la cabeza y desencolan del final de la cola, justo al revés que los métodos originales de Cola. Como se trata de añadir miembros nuevos, hacemos que herede de la interfaz Cola.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Si implementamos la interfaz ColaDoble en la clase Lista tendremos que implementar los cuatro métodos los dos definidos en Cola y los dos añadidos en ColaDoble Por tanto, al implementar ColaDoble estaremos implementando también Cola de forma automática.

# VARIABLES DE TIPO INTERFAZ

También se pueden crear variables cuyo tipo es una interfaz. Con ellas podremos referenciar cualquier objeto de cualquier clase que la implemente, a ella o a una subinterfaz.

Por ejemplo, podremos crear la variable de tipo Sonido.



Con ella es posible referenciar objetos de cualquier clase que implemente Sonido.



La sentencia:



Mostrará en la pantalla el mensaje ¡Miau!, ya que se ejecuta el método voz implementado en la clase Gato. Esa misma variable puede referenciar a un objeto Perro.



Con lo cual la misma sentencia:



Escribirá ¡Guau!, que es el mensaje implementado en la clase Perro.

Este es otro ejemplo de selección dinámica de métodos. La misma línea de código produce efectos distintos, ejecuta métodos distintos, dependiendo del objeto referenciado por la variable son. Se decide en tiempo de ejecución qué implementación concreta del método se va a ejecutar.

Es importante señalar que cuando usemos una variable de tipo Sonido para referenciar un objeto de la clase Perro, solo tendremos acceso a los métodos que pertenezcan a la clase interfaz Sonido y a los heredados de Object, pero no a los otros métodos implementados en la clase Perro

De esta forma, el objeto se manifestará exclusivamente como un objeto que emite sonido. Esto va a ocurrir con todas las variables de tipo interfaz. Volviendo al ejemplo de la interfaz Cola implementada en la clase Lista, si creamos una lista y la referenciamos con una variable de tipo Cola veremos que, con esa variable, solo podemos acceder a los métodos encolar() y desencolar() (aparte de los heredados de la clase Object que siempre son accesibles para cualquier objeto de cualquier clase) y no al resto de los miembros de la clase Lista. De ahí el nombre de interfaz a través de ella, veremos la lista como una cola pura.

Forma

Descripción generada automáticamente con confianza baja

Al escribir el punto después de la c, Eclipse despliega un menú con los métodos accesibles, entre los que solo figuran los de Cola y los heredados de Object.

# CLASES ANÓNIMAS

Lo más común es que las interfaces sean implementadas por clases, de las que luego se crearán diversos objetos. Sin embargo, hay ocasiones en que una determinada implementación de una interfaz es necesaria en un solo lugar. Cuando ocurre eso, no merece la pena definir una clase nueva para crear un solo objeto del que solo interesan las funcionalidades de la interfaz.

En estos casos, podemos crear sobre la marcha, en el mismo lugar del código donde se va a usar, un objeto de una clase sin nombre, es decir, anónima, en la que se implementan los métodos de la interfaz que nos interesa. Como no dispondremos de nombre para la clase, tampoco lo tendremos para el constructor. Por eso se usa uno con el nombre de la interfaz

Asimismo, el objeto será referenciado por una variable del tipo interfaz. Como ejemplo, crearemos una clase anónima que implemente la interfaz Sonido. Imaginemos que alguien ha encontrado un animal de una especie desconocida, que emite un extraño ruido. Mientras se identifica o no, para describir su sonido crearemos un objeto de una clase anónima que implementa la interfaz Sonido.

Texto

Descripción generada automáticamente

Por pantalla obtendremos el mensaje “¡Jajejijojuuuu!”. En realidad, no hemos definido ninguna clase. Lo que hemos hecho es crear un objeto sin clase, que nos permite describir el sonido que emite, aunque es costumbre hablar de clases anónimas. Además, todo esto se implementa en una línea de código ejecutable, no en una clase ni archivo aparte. Por eso se dice que su definición es local.

# ACCESO ENTRE MIEMBROS DE UNA INTERFAZ

Es importante destacar que, dentro de una interfaz, todos los métodos default y private tienen acceso entre ellos y, a su vez, pueden acceder a los métodos abstractos. No olvidemos que estos últimos son invocados a través de un objeto de alguna clase (aunque esta sea anónima), donde ya están implementados.

# SINTAXIS GENERAL

La sintaxis general de la definición de una interfaz tiene la siguiente forma:

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Si se omite tipoDeAcceso el acceso a la interfaz está restringido al paquete en el que está incluida. Si es public, la interfaz podrá ser importada desde otro paquete mediante una sentencia import.

Una clase puede implementar más de una interfaz, para lo cual deberá declararlas, separadas por comas, en el encabezamiento, e implementar todos los métodos abstractos de cada una de ellas. En general, la declaración de una clase que implementa una o más interfaces tiene la siguiente forma:

Gráfico

Descripción generada automáticamente con confianza baja

Cuando una clase implementa varias interfaces, puede haber algún método declarado en más de una de ellas con el mismo nombre y la misma lista de parámetros, el tipo devuelto tendría que ser también el mismo automáticamente si no, daría error de compilación. En este caso, en la clase se hará una única implementación del método.

En general, decimos que una clase implementa una interfaz si tiene implementados todos sus métodos abstractos, como hemos hecho en nuestros ejemplos. Sin embargo, se puede dejar alguno sin implementar con tal de que la clase se declare como abstracta.

Cuando una interfaz hereda de otras u otras, su definición tendrá la siguiente forma:

Texto

Descripción generada automáticamente

# UN PAR DE INTERFACES DE LA API

En programación hay operaciones tan necesarias y tan frecuentes, que merecen entrar a formar parte de las interfaces de la API. Una de esas operaciones es la de comparar valores para buscar u ordenar objetos

Con este fin, los desarrolladores de Java han implementado un par de interfaces, Comparable y Comparator. Las estudiamos aquí a modo de ejemplo. Además, las vamos a usar frecuentemente a partir de ahora.

## INTERFAZ COMPARABLE

Cuando queramos establecer un criterio de comparación natural o por defecto entre los objetos de una clase, haremos que implemente la interfaz Comparable que consta de un único método abstracto,

Logotipo

Descripción generada automáticamente con confianza media

Este método será el encargado de establecer un criterio de ordenación entre los objetos de la clase. Para comparar dos objetos ob 1 y ob 2 de una determinada clase que implemente Comparable escribiremos

Texto

Descripción generada automáticamente con confianza baja

Si en una ordenación el objeto ob 1 debe ir antes que el objeto ob 2 el método devolverá un número negativo si debe ir después, devolverá un número positivo, y si deben ser iguales a efectos de ordenación, devolverá un cero.

Un letrero de color blanco

Descripción generada automáticamente con confianza media

En dicho ejemplo, si observamos la implementación de compareTo(), llama la atención el cast Socio aplicado al parámetro ob como ya hicimos con el método equals() de la clase Object, que vimos en la unidad dedicada a las clases. La razón es que, para que la interfaz sea útil para cualquier clase de Java, su parámetro de entrada debe ser de la clase Object que es la más general posible.

Esto obliga a que, en el cuerpo de definición del método, haya que aplicarle un cast que le diga al compilador que, en realidad, es un objeto Socio y permita acceder así al atributo id.

En la implementación del método, el id del objeto que llama a compareTo es accesible directamente, ya que estamos accediendo al atributo desde el código del propio objeto this que hace la llamada. En cambio, ob es una referencia a un objeto externo, de tipo Object que se le pasa como parámetros, y para acceder a su id deberemos poner el cast delante.

En casos como este, en que se usa un atributo de tipo numérico en la comparación, hay una implementación mucho más sencilla:

Imagen que contiene Texto

Descripción generada automáticamente

Debemos tener en cuenta que el número positivo, o negativo, devuelto no tiene por qué ser 1 o -1. Lo que importa es si es positivo, negativo o 0.

Vamos a aplicar todo esto a la comparación de dos objetos Socio.

Texto

Descripción generada automáticamente

Se mostrará por pantalla un número positivo, ya que s1 iría después de s2 en una ordenación por número id. En cambio, si la tercera línea fuera



Aparecería un número negativo.

La interfaz Comparable ha sido creada por los desarrolladores de Java y la reconocen otras clases de la API Entre ellas, la clase Arrays donde se implementa el método sort que sirve para ordenar una tabla. Por ejemplo, si declaramos e inicializamos la tabla de números enteros



La sentencia



Ordena la tabla tabla por el orden natural de sus elementos que, en el caso de los número enteros, es el orden creciente.

Es importante tener en cuenta que el método sort no devuelve una copia ordenada de tabla, sino que ordena la tabla original. Si queremos mostrarla sin necesidad de un bucle que la recorra, podemos recurrir a otro método de la clase Arrays.

Texto

Descripción generada automáticamente

Que mostrará por pantalla



En este ejemplo hemos usado sort() en una tabla cuyos elementos son de un tipo primitivo, pero también sirve para ordenar una tabla de objetos de cualquier clase, con tal de que esta tenga implementada la interfaz Comparable. Usará el orden natural de la clase, es decir, el establecido por el método compareTo().

A continuación, se muestra un ejemplo vamos a crear una tabla de objetos Socio que, de paso, inicializaremos de una forma nueva:

Texto

Descripción generada automáticamente

Para ordenarlos utilizaremos el método sort() ya que Socio tiene implementada la interfaz Comparable que, como acabamos de ver, se basa en el atributo id creciente, escribiremos:



Podemos mostrar la tabla ya ordenada usando la sentencia:



donde, en lugar de toString() válido para tablas de tipos primitivos, hemos utilizado deepToString() que sirve para tablas de objetos. Este método llamará al método toString() que hayamos implementado en la clase Socio para mostrar los objetos individuales.

Se obtendrá por pantalla:

Texto

Descripción generada automáticamente

Las edades variarán con la fecha en que se ejecute el programa.

Muchas clases de la API de Java, como la clase String traen implementada la interfaz Comparable, con la cual podemos comparar sus objetos y ordenarlos con el método sort(). En particular, las cadenas se comparan siguiente el orden alfabético creciente.

Cuando se hacen búsquedas y ordenaciones de objetos de una clase definida por nosotros, además de la interfaz Comparable, debemos implementar el método equals(). Ambos identifican cuándo dos objetos son iguales, compareTo() devuelve 0 y equals() devuelve true. Para evitar conflictos, debemos asegurarnos de que ambos resultados sean consistentes, es decir, que identifiquen como iguales las mismas parejas de objetos.

## INTERFAZ COMPARATOR

La interfaz Comparable proporciona un criterio de ordenación natural, que es el que usan por defecto los distintos métodos de la API, como sort(). Pero es frecuente que tengamos que ordenar los objetos de la misma clase con distintos criterios en el mismo programa. Por ejemplo, puede ser que necesitemos un listado de objetos Socio por orden alfabético de nombres, o por edades, en sentido creciente o decreciente.

Para resolver este problema existe la interfaz Comparator definida también en la API Antes de usarla habrá que importarla con la sentencia:



Este interfaz tiene un único método abstracto:



Recibe como parámetros dos objectos que queremos comparar para determinar cuál va antes y cuál después en un proceso de ordenación. Devuelve un entero, que será negativo si ob1 va antes de ob2, positivo si va después y cero si son iguales Llamamos comparador a cualquier objeto de una clase que implemente la interfaz Comparator.

Necesitaremos una clase de comparadores distinta para cada criterio de comparación que queramos emplear con objetos de una clase determinada.

Pero, a diferencia de la interfaz Comparable, Comparator no se implementa en la clase de los objetos que queremos ordenar, en nuestro caso, Socio sino en una clase específica, cuyos objetos llamaremos comparadores. Por cada criterio de comparación adicional para la clase Socio, definiremos una nueva clase de comparadores.

Por ejemplo, si queremos ordenar una tabla de socios por orden creciente de edad manteniendo como ordena natural el atributo id, implementaremos una clase comparadora para el atributo edad.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Igual que ocurre con la interfaz Comparable, Comparator se ha diseñado para que sirva para comparar objetos de cualquier clase. Por eso los parámetros de entrada son de tipo Object. Esto obliga a usar el cast correspondiente, en este caso Socio.

Los métodos de ordenación de la API de Java, por ejemplo, el método sort(), están sobrecargados y admiten como parámetros adicional un comparador que les diga el criterio de ordenación que deben emplear. Como ejemplo vamos a reordenar la lista de socios por orden edades crecientes.

Empezamos por crear un objeto comparador de edades



Que pasaremos al método sort como argumento, junto con la tabla que queremos ordenar,



Para mostrar la tabla ordenada escribimos:



Obteniéndose por pantalla:

Texto

Descripción generada automáticamente

En vez de declarar la variable c podríamos haber creado el comparador en la propia llamada al método sort().



Otra opción, en caso de que se vaya a hacer la ordenación una sola vez, es crear una clase anónima en la llamada al método sort():

Texto

Descripción generada automáticamente

Cuando queramos una ordenación en sentido decreciente, basta cambiar el signo del valore devuelto en el método compare() o compareTo(). Sin embargo, la interfaz Comparator lleva implementado el método por defecto,



que, invocado por un objeto comparador, devuelve otro con el criterio de comparación invertido.

En clases más complejas, con más atributos, podrá haber más criterios posibles de ordenación Para cada uno de ellos habría que definir una clase comparadora específica.

La ordenación no es la única operación que precisa de un criterio de comparación. Cuando una tabla está ordenada, podemos hacer búsquedas rápidas, aprovechando el orden, por medio del método binarySearch() que está implementado en la clase Arrays igual que sort().

Para ello es indispensable que ambos métodos trabajen con el mismo criterio de ordenación. Por defecto, este criterio será el natural, implementado en la interfaz Comparable. Por ejemplo, si ordenamos una tabla t de números enteros por medio de la sentencia



La tabla quedará ordenada en orden natural (creciente) y podremos buscar el índice del valor 10 en la tabla escribiendo



Pero si la ordenamos en orden decreciente por medio de un objeto comparator de la clase ComparaEnterosInverso, implementado en la Actividad Resuelta 9.10 escribiremos:



Para buscar el valor 10 tendremos que pasar al método de búsqueda un argumento adicional con el mismo comparador:



En resumen, cuando en una aplicación tenemos que compara objetos de una clase con distintos criterios, tanto si es para ordenarlos como para hacer búsquedas binarias, definimos un criterio de ordenación natural, por defecto, a través de la interfaz Comparable y añadimos el resto de los criterios de ordenación con clases comparadoras, es decir, que implementan la interfaz Comparator.

Se llama criterio de orden a aquel con el cual se comparan dos valores u objetos en Java. Se usa para ordenar conjuntos de un tipo de datos, como tablas o colecciones (que se verán más adelantes) y para buscar elementos por métodos de búsqueda binaria.

Se llama criterio de ordenación natural de una clase al que se implementa a través de la interfaz Comparable, que usan por defecto los algoritmos de ordenación y búsqueda de la API.