PROGRAMACIÓN UD-7

CLASES

ÍNDICE

[INTRODUCCIÓN 3](#_Toc124774501)

[DEFINICIÓN DE UNA CLASE 3](#_Toc124774502)

[ATRIBUTOS 4](#_Toc124774503)

[A. INICIALIZACIÓN 4](#_Toc124774504)

[OBJETOS 4](#_Toc124774505)

[A. REFERENCIAS 4](#_Toc124774506)

[B. VARIABLES REFERENCIA 4](#_Toc124774507)

[C. OPERADOR NEW 5](#_Toc124774508)

[D. REFERENCIA null 5](#_Toc124774509)

[E. RECOLECTOR DE BASURA 6](#_Toc124774510)

[MÉTODOS 6](#_Toc124774511)

[A. ÁMBITO DE LAS VARIABLES Y ATRIBUTOS 6](#_Toc124774512)

[B. OCULTACIÓN DE ATRIBUTOS 7](#_Toc124774513)

[C. OBJETOS this 7](#_Toc124774514)

[ATRIBUTOS Y MÉTODOS ESTÁTICOS 8](#_Toc124774515)

[CONSTRUCTORES 8](#_Toc124774516)

[A. CONSTRUCTOR this() 9](#_Toc124774517)

# INTRODUCCIÓN

Hasta aquí hemos utilizado un paradigma de programación llamado programación estructurada, que emplea las estructuras de control (condicionales y bucles), junto a datos y funciones. Una de sus principales desventajas es que no existe un vínculo fuerte entre funciones y datos. Esto dificulta el tratamiento de problemas complejos.

Por eso, llegados a este punto, vamos a saltar a un nuevo paradigma, la programación orientada a objetos (en adelante POO), que amplía los horizontes de un programador, dotándolo de nuevas herramientas que facilitan la resolución de problemas complejos.

# DEFINICIÓN DE UNA CLASE

La POO se inspira en una abstracción del mundo real, en la que los objetos se clasifican en grupos.

Por ejemplo, todos los mamíferos de cuatro patas que dicen ¡guau!. Se engloban dentro del grupo de los perros.

Si observamos a Pepa, Paco y Miguel, vemos que los tres pertenecen al mismo grupo los tres son personas Pepa es una persona, Paco es una persona y Miguel también es una persona Todos pertenecen al grupo de las personas. En el argot de la POO, a cada uno de estos grupos se le denomina clase.

Podemos definir cada grupo o clase mediante las propiedades y comportamientos que presentan todos sus miembros. Una propiedad es un dato que conocemos de cada miembro del grupo, mientras que un comportamiento es algo que puede hacer.

Vamos a definir la clase persona mediante dos elementos:

* Propiedades. Un nombre, una edad y una estatura. Tanto Pepa como Paco como Miguel tienen un nombre, una edad y una estatura; son datos que en cada uno tendrá un valor distinto.
* Comportamientos. Pepa, Paco y Miguel, en realidad todas las personas, pueden saludar, crecer o cumplir años, por ejemplo.

Como vemos es posible definir una clase mediante un conjunto de propiedades y comportamientos. La sintaxis para definir una clase usa la palabra reservada class.

# ATRIBUTOS

Los datos que definen una clase se denominan atributos. Por ejemplo, la clase Vehículo puede definirse mediante los atributos matrícula, color, marca y modelo. Como hemos visto antes, nuestra clase Persona, dispone de los atributos nombre, edad y estatura.

El tipo específicado en tipo puede ser cualquier tipo primitivo o una clase.

## INICIALIZACIÓN

Se puede asignar un valor por defecto a los atributos de una clase, esto se realiza en la propia declaración.

En general, los atributos pueden cambiar durante la ejecución de un programa, salvo que sea declarados como constantes.

La inicialización de un atributo constante puede hacerse en su declaración o por medio de un constructor.

# OBJETOS

Los elementos que pertenecen a una clase se denominan instancias u objetos. Cada uno tiene sus propios valores para los atributos definidos en la clase.

## REFERENCIAS

El comportamiento de los objetos en la memoria del ordenador y sus operaciones elementales (creación, asignación y destrucción) son idénticos al de las tablas. Esto es debido a que ambos, objetos y tablas, utilizan las referencias. De hecho, las propias tablas se consideran en java como un tipo más de objetos.

Cualquier dato almacenado en la memoria ocupará, dependiendo de su tamaño, una serie de bloques consecutivos, y puede ser identificado mediante la dirección del primero de ellos. A esta primera dirección de memoria que identifica un objeto se le denomina en Java referencia.

## VARIABLES REFERENCIA

Antes de construir un objeto necesitamos declarar una variable cuyo tipo sea su clase. La declaración sigue las mismas reglas que las variables de tipo primitivo.

La diferencia entre una variable de tipo primitivo y una variable de tipo referencia es que mientras una variable de tipo primitivo almacena directamente un valor, una variable del tipo clase almacena la referencia de un objeto.

## OPERADOR NEW

La forma de crear objetos, como en las tablas, es mediante el operador new.

El operador new primero busca en memoria un hueco disponible donde construir el objeto. Este, dependiendo de su tamaño, ocupará cierto número consecutivo de bloques de memoria. Por último, devuelve la referencia del objeto recién creado, que se asigna a la variable creada.

El tamaño de una clase depende del tipo y la cantidad de atributos que esta contenga.

A la hora de trabajar con referencias, es bastante más sencillo pensar en ellas como flechas que se dirigen desde la variable hacia el objeto.

En el momento en el que disponemos de un objeto, podemos acceder a sus atributos mediante el nombre de la variable seguido de un punto.

Es importante comprender que podemos acceder al mismo objeto mediante distintas variables que almacenan la misma referencia.

El mecanismo en el que varias variables comparten la misma referencia es aprovechado por la clase String para ahorrar espacio en textos usados frecuentemente, ya que Java se encarga por su cuenta de que todas las variables a las que se les han asignado idéntico literal cadena compartan su referencia.

El hecho de que se compartan las referencias de un mismo literal cadena es la causa de que la clase sea inmutable.

## REFERENCIA null

El valor literal “null” es una referencia nula. Es decir, es una referencia a ningún bloque de memoria. Cuando declaramos una variable referencia se inicializa por defecto a “null”.

Hay que tener mucho cuidado de no intentar acceder a los miembros de una referencia nula, ya que se produce un error que termina la ejecución de forma inesperada.

El “null” se puede asignar a cualquier variable referencia.

## RECOLECTOR DE BASURA

Existen tres formas de conseguir que un objeto no esté referenciado:

1. Creándolo y no asignándolo a ninguna variable.
2. Asignándoles el valor “null” a todas las variables que contenían una referencia a un objeto.
3. Asignando el objeto a una variable distinta.

En todos los casos, el objeto se queda perdido en memoria, es decir, no existe forma de acceder a él. Sin embargo, está ocupando memoria. Si este comportamiento se repite demasiado, fortuita o malintencionadamente es posible que se agote toda la memoria libre disponible, lo que impediría el normal funcionamiento del ordenador.

Para evitar este problema, Java dispone de un mecanismo llamado, recolector de basura, “garbage collector”, que se ejecuta de vez en cuando de forma transparente al usuario, y se encarga de comprobar uno a uno, todos los objetos de la memoria. Si alguno de ellos no estuviera referenciado por ninguna variable, se destruye, liberando la memoria que ocupa.

# MÉTODOS

Hemos declarado clases con atributos, pero también disponen de comportamientos. En el argot de la POO, a los comportamientos u operaciones que pueden realizar los objetos de una clase se les denomina **métodos**. Por ejemplo, las personas son capaces de realizar operaciones como saludar, cumplir años, crecer, etc.

Los métodos no son más que funciones que se implementan dentro de una clase.

Tanto a los atributos como a los métodos de una clase se les llama de forma genérica miembros. De esta forma, al hablar de miembros de una clase, hacemos referencia a los atributos y métodos declarados en su definición.

Los métodos de una clase tienen acceso a las siguientes variables: variables locales declaradas dentro del método, parámetros de entrada y atributos de la clase. Así mismo, tiene acceso a los demás métodos de la clase.

## ÁMBITO DE LAS VARIABLES Y ATRIBUTOS

El ámbito de una variable define en qué lugar puede usarse y coincide con el bloque en el que se declara la variable que puede ser:

* + El bloque de una estructura de control que dará lugar a variables de bloque.
  + Una función o método que dará lugar a variables locales.

Con la POO, aparece un nuevo ámbito: la clase que dará lugar a variables de la clase.

Un ámbito puede contener a otros ámbitos, formando una estructura jerárquica. Por ejemplo, una clase puede contener dos métodos, y estos, distintos bloques de, por ejemplo, una estructura while o if.

Una variable puede utilizarse en el ámbito o bloque en el que se declara, que incluye sus bloques internos. Sin embargo, no ocurre lo contrario: una variable no podrá utilizarse en el ámbito padre del bloque en el que se declara.

Por ejemplo, un atributo puede emplearse dentro de un método, y una variable local dentro del bloque de una estructura de control de un método. Pero, en cambio, no podemos usar una variable local fuera de su método, ni una variable de bloque fuera de él.

## OCULTACIÓN DE ATRIBUTOS

Dos variables declaradas en ámbitos anidados no pueden tener el mismo identificador, ya que esto genera un error. Sin embargo, existe una excepción cuando una variable local en un método tiene el mismo identificado que un atributo de la clase. En este caso, dentro del método, la variable local tiene prioridad sobre el atributo, es decir, que al utilizar el identificador se accede a la variable local y no al atributo. En la jerga de la POO se dice que la variable local oculta al atributo.

## OBJETOS this

La palabra reservada this permite utilizar un atributo incluso cuando ha sido ocultado por la variable local. De igual manera que cada uno se refiere a sí mismo como yo, aunque tengamos un nombre que los demás utilizan para identificarnos, las clases se refieren a sí mismas como this, que es una referencia al objeto actual y funciona como una especie de yo para las clases.

Al escribir this en el ámbito de una clase se interpreta como la propia clase, y permite acceder a los atributos aunque se encuentren ocultos.

# ATRIBUTOS Y MÉTODOS ESTÁTICOS

Un atributo estático, también llamado atributo de la clase, es aquel del que no existe una copia en cada objeto. Todos los objetos de una misma clase comparten su valor.

Un atributo estático se inicializará en el momento de cargar la clase en memoria; esto ocurre cuando se declara alguna variable del tipo de la clase o cuando se crea un primer objeto de dicha clase.

También podemos declarar métodos estáticos. Son aquellos que no requieren de ningún objeto para ejecutarse y, por tanto, no pueden utilizar ningún atributo que no sea estático. En el caso de que lo intente se producirá un error.

La forma de invocar un método estático es mediante el nombre de la clase.

Por otra parte, desde un método estático sólo se pude invocar directamente métodos y atributos estáticos.

No obstante, dentro de un método estático se pueden crear objetos de cualquier clase, incluida la suya propia, y desde él invocar miembros no estáticos definidos en esta clase.

# CONSTRUCTORES

Es un método especial que debe tener el mismo nombre que la clase, se define sin tipo devuelto (ni siquiera void), y se ejecuta inmediatamente después de crear el objeto.

El principal cometido de un constructor es asignar valores a los atributos, aunque también se puede utilizar para otros fines como crear tablas, mostrar cualquier tipo de información, crear otros objetos que necesitemos, etc.

Al constructor como cualquier otro método, se le pueden pasar parámetros y se puede sobrecargar.

La llamada al constructor con los valores de los parámetros de entrada se hace por medio del operador new.

Los atributos declarados como final también se pueden inicializar pasando sus valores como parámetros al constructor; no es necesario en el sitio donde se declaran.

A la hora de sobrecargar un método tenemos que asegurarnos de que se puede distinguir entre las distintas versiones mediante el número o el tipo de parámetros de entrada. La sobrecarga de constructores es útil cuando necesitamos inicializar objetos de varias formas.

Cuando en una clase no se implementa ningún constructor, Java se encarga de crear uno que se denomina constructor por defecto. Este no usa parámetros de entrada e inicializa los atributos a cero, false o null según el tipo si no están ya inicializados en su declaración.

No obstante, es conveniente implementar los constructores y no dejarlo en manos de Java. En cuanto se implementa un constructor en una clase, el constructor, por defecto, deja de estar disponible.

## CONSTRUCTOR this()

Cuando una clase dispone de un conjunto de constructores sobrecargados, es posible que un constructor invoque a otro y así reutilice su funcionalidad. Para eso se usa el constructor genérico this(), en lugar del constructor por su nombre.

La forma de distinguir los distintos constructores, igual que en cualquier método sobrecargado, es mediante el número y el tipo de los parámetros de entrada.

Tenemos que tener presente que, en caso de utilizar this(), tiene que ser siempre la primera instrucción de un constructor; en otro caso se producirá un error.

# PAQUETES

En Java es importante controlar la accesibilidad de unas clases desde otras por razones de seguridad y eficiencia. Esto se consigue mediante paquetes, que son contenedores que permiten guardar clases en compartimentos separados, de modo que podamos decidir, por medio de la importación, qué clases son accesibles y cómo se accede a ellas desde una clase que estemos implementando.

Todas las clases están dentro de algún paquete que, a su vez, pueden estar anidados unos dentro de otros. Se considera que una clase que pertenece a un paquete, que a su vez está dentro de otro, sólo pertenece al primero, pero no al segundo.

Archivo fuente de Java es un archivo de texto con extensión. Java, que se guarda en un paquete y que contiene los siguientes elementos:

* Una sentencia donde se especifica el paquete al que pertenece, que empieza con la palabra clave ***package*** seguida del nombre del paquete.
* Una serie opcional de sentencias de importación, con la palabra reservada **import**, que permite importar clases definidas en otros paquetes.
* La definición de una o más clases, de las cuales sólo una puede ser declarada pública, por medio del modificado de acceso **public**. De todas formas, es recomendable que en cada archivo fuente se defina una sola clase, que debe tener el nombre del archivo.

# MODIFICADORES DE ACCESO

Una clase será visible por otra, o no, dependiendo de si se ubican en el mismo paquete y de los modificadores de acceso que utilice. Estos modifican su visibilidad, permitiendo que se muestre u oculte.

De igual manera que podemos modificar la visibilidad entre las clases, es posible modificar la visibilidad entre los miembros de distintas clases, es decir, qué atributos y métodos son visibles para otras clases.

## MODIFICADORES DE ACCESO PARA CLASES

Debido a la estructura de clases, organizadas en paquetes, que utiliza Java, dos clases cualesquiera pueden definirse de las siguientes formas:

* + Clases vecinas: pertenecen al mismo paquete.
  + Clases externas: pertenecen a paquetes distintos.

Existen varios grados de visibilidad:

* + Visibilidad por defecto: Cuando se define una clase sin ningún modificador de acceso hace que sólo sea visible desde clases vecinas.
  + Visibilidad total: Cuando se utiliza el modificador public la clase además de ser visible por sus clases vecinas, lo será por las clases externas previa importación.

## MODIFICADORES DE ACCESO PARA MIEMBROS

De igual manera que es posible modificar la visibilidad de una clase, podemos regular la visibilidad de sus miembros. Que un atributo sea visible significa que podemos acceder a él, tanto para leer como para modificarlo. Que un método sea visible significa que puede ser invocado.

Para que un miembro sea visible, es indispensable que su clase también lo sea. Es evidente que, si no podemos acceder a una clase, no existe forma de acceder a sus miembros.

Debemos destacar que cualquier miembro es siempre visible dentro de su propia clase, indistintamente del modificador de acceso que utilicemos. Es decir, desde dentro de la definición de una clase siempre tendremos acceso a todos sus atributos y podremos invocar cualquiera de sus métodos.

* + Visibilidad por defecto. Cuando queramos acceder a miembros de otra clase hay diversos grados de visibilidad. La visibilidad por defecto es aquella que se aplica a miembros declarados sin ningún modificador de acceso.

La visibilidad por defecto hace que un miembro sea visible desde las clases vecinas, pero invisible desde clases externas.

* + Modificador de acceso private y public. Con el modificador de acceso private obtenemos una visibilidad más restrictiva que por defecto, ya que impide el acceso para las clases vecinas. Un miembro, ya sea un atributo o un método, declarado privado es invisible desde fuera de la clase.

En cambio, public hace que un miembro sea visible incluso desde clases externas previa importación. Otorga visibilidad total.

Usaremos private cuando queramos controlar los cambios de un atributo o cuando deseamos que no se conozca directamente su valor, o bien cuando queremos que un método sea invocado desde otros métodos de la clase, pero no fuera de ella.

El acceso a los miembros privados deberá hacerse a través de algún método public de la misma clase.

## MÉTODOS GET/SET

Un atributo público puede ser modificado desde cualquier clase, lo que a veces tiene sus inconvenientes, ya que es imposible controlar los valores asignados, que pueden no tener sentido.

Por este motivo, existe una convención en la comunidad de programadores que consiste en ocultar atributos y, en su lugar, crear dos métodos públicos: el primero, habitualmente llamado set, permite asignar un valor al atributo, controlando el rango válido de valores. El segundo, habitualmente llamado get, devuelve el atributo, lo que posibilita conocer su valor. Los métodos set/get hacen que un atributo no visible se comporte como si lo fuera.

Las ventajas de utilizar métodos set/get son que la implementación de la clase se encapsula, ocultando los detalles y, por otro lado, permite controlar qué atributos son accesibles para lectura y cuáles para escritura, así como los valores asignados.

# ENUMERADOS

Los tipos enumerados sirven para definir grupos de constantes como posibles valores de una variable.

Se definen de forma parecida a una clase pero utilizando la palabra clave enum y no class. En un programa se accede a sus valores de la forma nombreEnumerado.valorEnumerado (por ejemplo, DiaDeLaSemana.LUNES).

Un tipo enumerado se puede implementar en un archivo aparte, normalmente dentro del mismo paquete, aunque no es obligatorio, como si fuera una clase o bien dentro de la definición donde se va a usar.

Los tipos enumerados se pueden definir en paquetes distintos a donde se vayan a usar. En este caso, habrá que definirlos public e importarlos igual que si fueran clases.

# WRAPPERS

También conocidos como envoltorios son clases que internamente contienen un dato de tipo primitivo, lo que proporciona una forma de trabajar con estos como objetos. Cada tipo primitivo tiene su correspondiente clase envoltorio: Integer para los int, Double para los double, Character para los char, etc.