

# Fundamentos de la Programación Orientada a Objetos.

---



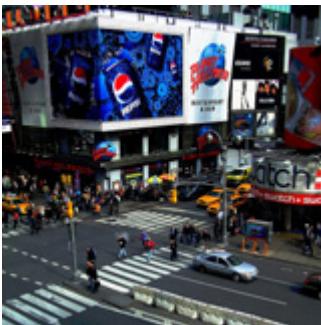
## Objetivos

En esta unidad se introducirá al alumnado en el concepto de Programación Orientada a Objetos. Se definirán conceptos tales como Abstracción, Encapsulación, Herencia y Polimorfismo.

Seguidamente se definirán qué son las clases y los objetos, para pasar a detallar la utilización de los mismos, así como de sus métodos. En el próximo tema veremos cómo crear clases y objetos en Java.

# 1.- Introducción.

---



Si nos paramos a observar el mundo que nos rodea, podemos apreciar que casi todo está formado por **objetos**. Existen coches, edificios, sillas, mesas, semáforos, ascensores e incluso personas o animales. Todos ellos pueden ser considerados objetos, con una serie de **características** y **comportamientos**. Por ejemplo, existen coches de diferentes marcas, colores, etc. y pueden acelerar, frenar, girar, etc., o las personas tenemos diferente color de pelo, ojos, altura y peso y podemos nacer, crecer, comer, dormir, etc.

Los programas son el resultado de la búsqueda y obtención de una solución para un problema del mundo real. Pero ¿en qué medida los programas están organizados de la misma manera que el problema que tratan de solucionar? La respuesta es que muchas veces los programas se ajustan más a los términos del sistema en el que se ejecutarán que a los del propio problema.

Si redactamos los programas utilizando los mismos términos de nuestro mundo real, es decir, utilizando objetos, y no los términos del sistema o computadora donde se vaya a ejecutar, conseguiremos que éstos sean más legibles y, por tanto, más fáciles de modificar.

Esto es precisamente lo que pretende la **Programación Orientada a Objetos (POO)**, en inglés **OOP (Object Oriented Programming)**, establecer una serie de técnicas que permitan trasladar los problemas del mundo real a nuestro sistema informático.

## 2.- Fundamentos de la Programación Orientada a Objetos.

Dentro de las distintas formas de hacer las cosas en programación, distinguimos dos paradigmas fundamentales:

- ✓ **Programación Estructurada**, se crean funciones y procedimientos que definen las acciones a realizar, y que posteriormente forman los programas.
- ✓ **Programación Orientada a Objetos**, considera los programas en términos de **objetos** y todo gira alrededor de ellos.



Pero ¿en qué consisten realmente estos paradigmas? Veamos estos dos modelos de programación con más detenimiento. Inicialmente se programaba aplicando las técnicas de programación tradicional, también conocidas como **Programación Estructurada**. El problema se descomponía en unidades más pequeñas hasta llegar a acciones más simples y fáciles de codificar.

Por ejemplo, en la resolución de una ecuación de primer grado, lo que hacemos es descomponer el problema en acciones más pequeñas o pasos diferenciados:

- ✓ Pedir valor de los coeficientes.
- ✓ Calcular el valor de la incógnita.
- ✓ Mostrar el resultado.

Si nos damos cuenta, esta serie de acciones o pasos diferenciados no son otra cosa que verbos; por ejemplo el verbo pedir, calcular, mostrar, etc.

Sin embargo, la **Programación Orientada a Objetos** aplica de otra forma diferente la técnica de programación "divide y vencerás". Este paradigma surge en un intento de salvar las dificultades que, de forma innata, posee el software. Para ello lo que hace es descomponer, en lugar de en acciones, en objetos. El principal objetivo sigue siendo descomponer el problema en problemas más pequeños, que sean fáciles de manejar y mantener, fijándonos en cuál es el escenario del problema e intentando reflejarlo en nuestro programa. O sea, se trata de trasladar la visión del mundo real a nuestros programas. Por este motivo se dice que la Programación Orientada a Objetos aborda los problemas de una forma más **natural**, entendiendo como natural que está más en contacto con el mundo que nos rodea.

La Programación Estructurada se centra en el conjunto de acciones a realizar en un programa, haciendo una división de procesos y datos. La Programación Orientada a Objetos se centra en la relación que existe entre los datos y las acciones a realizar con ellos, y los encierra dentro del concepto de objeto, tratando de realizar una abstracción lo más cercana al mundo real.



### Autoevaluación

Relaciona el término con su definición, escribiendo el número asociado a la definición en el hueco correspondiente.

#### Ejercicio de relacionar

Paradigma	Relación	Definición
1	2	3

Paradigma	Relación	Definición
Programación Orientada a Objetos.		<p>1. Maneja funciones y procedimientos que definen las acciones a realizar.</p>
Programación Estructurada.		<p>2. Representa las entidades del mundo real mediante componentes de la aplicación.</p>

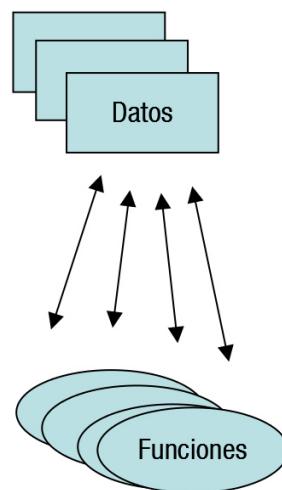
Enviar

## 2.1.- Conceptos.

Para entender mejor la filosofía de orientación a objetos veamos algunas características que la diferencian de las técnicas de programación tradicional.

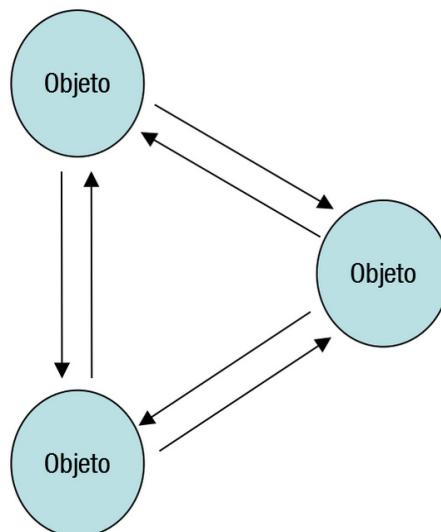
En la Programación **Estructurada**, el programa estaba compuesto por un conjunto de **datos y funciones** "globales". El término global significaba que eran accesibles por todo el programa, pudiendo ser llamados en cualquier ubicación de la aplicación. Dentro de las funciones se situaban las instrucciones del programa que manipulaban los datos. Funciones y datos se encontraban **separados y totalmente independientes**. Esto ocasionaba dos problemas principales:

- ✓ Los programas se creaban y estructuraban de acuerdo con la arquitectura del ordenador en el que se tenían que ejecutar.
- ✓ Al estar separados los datos de las funciones, éstos eran visibles en toda la aplicación. Ello ocasionaba que cualquier modificación en los datos podía requerir la modificación en todas las funciones del programa, en correspondencia con los cambios en los datos.



En la Programación **Orientada a Objetos** la situación es diferente. La utilización de **objetos** permite un mayor nivel de **abstracción** que con la Programación Estructurada, y ofrece las siguientes diferencias con respecto a ésta:

- ✓ El programador organiza su programa en **objetos**, que son **representaciones del mundo real** que están más cercanas a la forma de pensar de la gente.
- ✓ Los datos, junto con las funciones que los manipulan, son parte interna de los objetos y no están accesibles al resto de los objetos. Por tanto, los cambios en los datos de un objeto sólo afectan a las funciones definidas para ese objeto, pero no al resto de la aplicación.



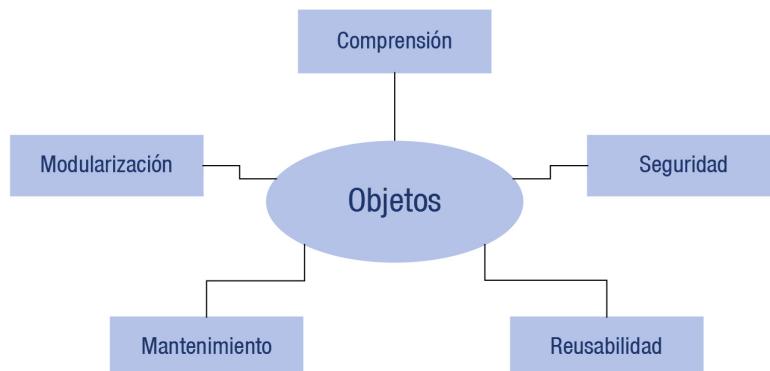
Todos los programas escritos bajo el paradigma orientado a Objetos se pueden escribir igualmente

mediante la Programación Estructurada. Sin embargo, la Programación Orientada a Objetos es la que mayor facilidad presenta para el desarrollo de programas basados en interfaces gráficas.

## 2.2.- Beneficios.

Según lo que hemos visto hasta ahora, un objeto es **cualquier entidad que podemos ver o apreciar**. El concepto fundamental de la Programación Orientada a Objetos son, precisamente, los **objetos**. Pero ¿qué beneficios aporta la utilización de objetos? Fundamentalmente la posibilidad de representar el problema en términos del mundo real, que como hemos dicho están más cercanos a nuestra forma de pensar, pero existen otra serie de ventajas como las siguientes:

- ✓ **Comprendión.** Los conceptos del espacio del problema se hayan reflejados en el código del programa, por lo que la mera lectura del código nos describe la solución del problema en el mundo real.
- ✓ **Modularidad.** Facilita la modularidad del código, al estar las definiciones de objetos en módulos o archivos independientes, hace que las aplicaciones estén mejor organizadas y sean más fáciles de entender.
- ✓ **Fácil mantenimiento.** Cualquier modificación en las acciones queda automáticamente reflejada en los datos, ya que ambos están estrechamente relacionados. Esto hace que el mantenimiento de las aplicaciones, así como su corrección y modificación sea mucho más fácil. Por ejemplo, podemos querer utilizar un algoritmo más rápido, sin tener que cambiar el programa principal. Por otra parte, al estar las aplicaciones mejor organizadas, es más fácil localizar cualquier elemento que se quiera modificar y/o corregir. Esto es importante ya que se estima que los mayores costes de software no están en el proceso de desarrollo en sí, sino en el mantenimiento posterior de ese software a lo largo de su vida útil.
- ✓ **Seguridad.** La probabilidad de cometer errores se ve reducida, ya que no podemos modificar los datos de un objeto directamente, sino que debemos hacerlo mediante las acciones definidas para ese objeto. Imaginemos un objeto lavadora. Se compone de un motor, tambor, cables, tubos, etc. Para usar una lavadora no se nos ocurre abrirla y empezar a manipular esos elementos, ya que lo más probable es que se estropie. En lugar de eso utilizamos los programas de lavado establecidos. Pues algo parecido con los objetos, no podemos manipularlos internamente, sólo utilizar las acciones que para ellos hay definidas.
- ✓ **Reusabilidad.** Los objetos se definen como entidades reutilizables, es decir, que los programas que trabajan con las mismas estructuras de información, pueden reutilizar las definiciones de objetos empleadas en otros programas, e incluso las acciones definidas sobre ellos. Por ejemplo, podemos crear la definición de un objeto de tipo **persona** para una aplicación de negocios y deseamos construir a continuación otra aplicación, digamos de educación, en donde utilizamos también personas, no es necesario crear de nuevo el objeto, sino que por medio de la reusabilidad podemos utilizar el tipo de objeto **persona** previamente definido.

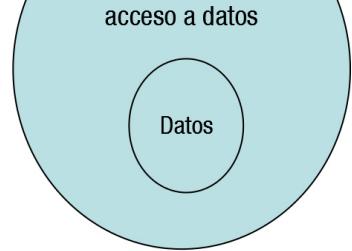


## 2.3.- Características.

Cuando hablamos de Programación Orientada a Objetos, existen una serie de características que se deben cumplir. Cualquier lenguaje de programación orientado a objetos las debe contemplar. Las características más importantes del paradigma de la programación orientada a objetos son:

- ✓ **Abstracción.** Es el proceso por el cual definimos las características más importantes de un objeto, sin preocuparnos de cómo se escribirán en el código del programa, simplemente lo definimos de forma general. En la Programación Orientada a Objetos la herramienta más importante para soportar la abstracción es la **clase**. Básicamente, una clase es un tipo de dato que agrupa las características comunes de un conjunto de objetos. Poder ver los objetos del mundo real que deseamos trasladar a nuestros programas, en términos abstractos, resulta de gran utilidad para un buen diseño del software, ya que nos ayuda a comprender mejor el problema y a tener una visión global de todo el conjunto. Por ejemplo, si pensamos en una clase **Vehículo** que agrupa las características comunes de todos ellos, a partir de dicha clase podríamos crear objetos como **Coche** y **Camión**. Entonces se dice que **Vehículo** es una abstracción de **Coche** y de **Camión**.
- ✓ **Modularidad.** Una vez que hemos representado el escenario del problema en nuestra aplicación, tenemos como resultado un conjunto de objetos software a utilizar. Este conjunto de objetos se crean a partir de una o varias clases. Cada clase se encuentra en un archivo diferente, por lo que la modularidad nos permite modificar las características de la clase que define un objeto, sin que esto afecte al resto de clases de la aplicación.
- ✓ **Encapsulación.** También llamada "**ocultamiento de la información**". La **encapsulación** o **encapsulamiento** es el mecanismo básico para ocultar la información de las partes internas de un objeto a los demás objetos de la aplicación. Con la encapsulación un objeto puede ocultar la información que contiene al mundo exterior, o bien restringir el acceso a la misma para evitar ser manipulado de forma inadecuada. Por ejemplo, pensemos en un programa con dos objetos, un objeto **Persona** y otro **Coche**. **Persona** se comunica con el objeto **Coche** para llegar a su destino, utilizando para ello las acciones que **Coche** tenga definidas como por ejemplo conducir. Es decir, **Persona** utiliza **Coche** pero no sabe cómo funciona internamente, sólo sabe utilizar sus métodos o acciones.
- ✓ **Jerarquía.** Mediante esta propiedad podemos definir relaciones de jerarquías entre clases y objetos. Las dos jerarquías más importantes son la jerarquía "**es un**" llamada **generalización** o **especialización** y la jerarquía "**es parte de**", llamada **agregación**. Conviene detallar algunos aspectos:
  - ◆ La generalización o especialización, también conocida como **herencia**, permite crear una clase nueva en términos de una clase ya existente (herencia simple) o de varias clases ya existentes (herencia múltiple). Por ejemplo, podemos crear la clase **CochedeCarreras** a partir de la clase **Coche**, y así sólo tendremos que definir las nuevas características que tenga.
  - ◆ La agregación, también conocida como **inclusión**, permite agrupar objetos relacionados entre sí dentro de una clase. Así, un **Coche** está formado por **Motor**, **Ruedas**, **Frenos** y **Ventanas**. Se dice que **Coche** es una agregación y **Motor**, **Ruedas**, **Frenos** y **Ventanas** son agregados de **Coche**.
- ✓ **Polimorfismo.** Esta propiedad indica la capacidad de que varias clases creadas a partir de una antecesora realicen una misma acción de forma diferente. Por ejemplo, pensemos en la clase **Animal** y la acción de expresarse. Nos encontramos que cada tipo de **Animal** puede hacerlo de manera distinta, los **Perros** ladran, los **Gatos** maullan, las **Personas** hablamos, etc. Dicho de otra manera, el polimorfismo indica la posibilidad de tomar un objeto (de tipo **Animal**, por ejemplo), e indicarle que realice la acción de expresarse, esta acción será diferente según el tipo de mamífero del que se trate.





### 3.- Características de los objetos.

Al principio de la unidad veíamos que el mundo real está compuesto de objetos, y podemos considerar objetos casi cualquier cosa que podemos ver y sentir. Cuando escribimos un programa en un lenguaje orientado a objetos, debemos identificar cada una de las partes del problema con objetos presentes en el mundo real, para luego trasladarlos al modelo computacional que estamos creando.

En este contexto, un objeto de software es una representación de un objeto del mundo real, compuesto de una serie de características y un comportamiento específico. Pero ¿qué es más concretamente un objeto en Programación Orientada a Objetos? Veámoslo.

**Un objeto es un conjunto de datos con las operaciones definidas para ellos.** Los objetos tienen un **estado** y un **comportamiento**.

Por tanto, estudiando los objetos que están presentes en un problema podemos dar con la solución a dicho problema. Los objetos tienen unas características fundamentales que los distinguen:

- ✓ **Identidad.** Es la característica que permite diferenciar un objeto de otro. De esta manera, aunque dos objetos sean exactamente iguales en sus atributos, son distintos entre sí. Puede ser una dirección de memoria, el nombre del objeto o cualquier otro elemento que utilice el lenguaje para distinguirlos. Por ejemplo, dos vehículos que hayan salido de la misma cadena de fabricación y sean iguales aparentemente, son distintos porque tienen un código que los identifica.
- ✓ **Estado.** El estado de un objeto viene determinado por una serie de parámetros o atributos que lo describen, y los valores de éstos. Por ejemplo, si tenemos un objeto **Coche**, el estado estaría definido por atributos como **Marca, Modelo, Color, Cilindrada**, etc.
- ✓ **Comportamiento.** Son las acciones que se pueden realizar sobre el objeto. En otras palabras, son los métodos o procedimientos que realiza el objeto. Siguiendo con el ejemplo del objeto **Coche**, el comportamiento serían acciones como: **arrancar(), parar(), acelerar(), frenar()**, etc.

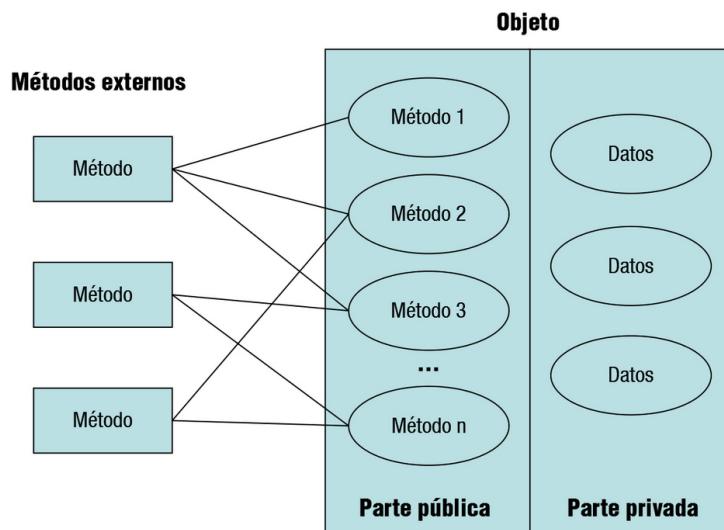
## 3.1.- Propiedades y métodos

Como acabamos de ver todo objeto tiene un estado y un comportamiento. Concretando un poco más, las partes de un objeto son:

- ✓ Campos, Atributos o **Propiedades**: Parte del objeto que almacena los datos. También se les denomina Variables **Miembro**. Estos datos pueden ser de cualquier tipo primitivo (`boolean`, `char`, `int`, `double`, etc) o ser su vez ser otro objeto. Por ejemplo, un objeto de la clase `Coche` puede tener un objeto de la clase `Ruedas`.
- ✓ **Métodos** o Funciones Miembro: Parte del objeto que lleva a cabo las operaciones sobre los atributos definidos para ese objeto.

La idea principal es que el objeto reúne en una sola entidad los datos y las operaciones, y para acceder a los datos privados del objeto debemos utilizar los métodos que hay definidos para ese objeto.

La única forma de **manipular** la información del **objeto** es a través de sus **métodos**. Es decir, si queremos saber el valor de algún atributo, tenemos que utilizar el método que nos muestre el valor de ese atributo. De esta forma, evitamos que métodos externos puedan alterar los datos del objeto de manera inadecuada. Se dice que los datos y los métodos están **encapsulados** dentro del objeto.



## 3.2.- Interacción entre objetos.

Dentro de un programa los objetos se comunican llamando unos a otros a sus métodos. Los métodos están dentro de los objetos y describen el comportamiento de un objeto cuando recibe una llamada a uno de sus métodos. En otras palabras, cuando un objeto, **objeto1**, quiere actuar sobre otro, **objeto2**, tiene que ejecutar uno de sus métodos. Entonces se dice que el **objeto2** recibe un mensaje del **objeto1**.

Un **mensaje** es la acción que realiza un objeto.

Un **método** es la función o procedimiento al que se llama para actuar sobre un objeto.

Los distintos mensajes que puede recibir un objeto o a los que puede responder reciben el nombre de protocolo de ese objeto.

El proceso de interacción entre objetos se suele resumir diciendo que se ha "enviado un mensaje" (hecho una petición) a un objeto, y el objeto determina "qué hacer con el mensaje" (ejecuta el código del método). Cuando se ejecuta un programa se producen las siguientes acciones:

- ✓ **Creación** de los objetos a medida que se necesitan.
- ✓ **Comunicación** entre los objetos mediante el envío de mensajes unos a otros, o el usuario a los objetos.
- ✓ **Eliminación** de los objetos cuando no son necesarios para dejar espacio libre en la memoria del ordenador.

Los objetos se pueden comunicar entre ellos invocando a los métodos de los otros objetos.



### Autoevaluación

Cuando un objeto, **objeto1**, ejecuta un método de otro, **objeto2**, se dice que el **objeto2** le ha mandado un mensaje al **objeto1**.

- Verdadero  Falso

## 3.3.- Ejemplo

Recuerda que entre las características fundamentales de un objeto se encontraban la **identidad** (los objetos son únicos y por tanto distinguibles entre sí, aunque pueda haber objetos exactamente iguales), un **estado** (los atributos que describen al objeto y los valores que tienen en cada momento) y un determinado **comportamiento** (acciones que se pueden realizar sobre el objeto).

Algunos ejemplos de objetos que podríamos imaginar podrían ser:

- ✓ Un coche de color rojo, marca SEAT, modelo Toledo, del año 2003. En este ejemplo tenemos una serie de atributos, como el color (en este caso rojo), la marca, el modelo, el año, etc. Así mismo también podríamos imaginar determinadas características como la cantidad de combustible que le queda, o el número de kilómetros recorridos hasta el momento.
- ✓ Un coche de color amarillo, marca Opel, modelo Astra, del año 2002.
- ✓ Otro coche de color amarillo, marca Opel, modelo Astra y también del año 2002. Se trataría de otro objeto con las mismas propiedades que el anterior, pero sería un segundo objeto.
- ✓ Un cocodrilo de cuatro metros de longitud y de veinte años de edad.
- ✓ Un círculo de radio 2 centímetros, con centro en las coordenadas (0,0) y relleno de color amarillo.
- ✓ Un círculo de radio 3 centímetros, con centro en las coordenadas (1,2) y relleno de color verde.

Si observas los ejemplos anteriores podrás distinguir sin demasiada dificultad al menos tres familias de objetos diferentes, que no tienen nada que ver una con otra:

- ✓ Los coches.
- ✓ Los círculos.
- ✓ Los cocodrilos.

Es de suponer entonces que cada objeto tendrá determinadas posibilidades de **comportamiento (acciones)** dependiendo de la familia a la que pertenezcan. Por ejemplo, en el caso de los **coches** podríamos imaginar acciones como: arrancar, frenar, acelerar, cambiar de marcha, etc. En el caso de los **cocodrilos** podrías imaginar otras acciones como: desplazarse, comer, dormir, cazar, etc. Para el caso del **círculo** se podrían plantear acciones como: cálculo de la superficie del círculo, cálculo de la longitud de la circunferencia que lo rodea, etc.

Por otro lado, también podrías imaginar algunos **atributos** cuyos valores podrían ir cambiando en función de las acciones que se realizaran sobre el objeto: ubicación del coche (coordenadas), velocidad instantánea, kilómetros recorridos, velocidad media, cantidad de combustible en el depósito, etc. En el caso de los cocodrilos podrías imaginar otros atributos como: peso actual, el número de dientes actuales (irá perdiendo algunos a lo largo de su vida), el número de presas que ha cazado hasta el momento, etc.

Como puedes ver, un **objeto** puede ser cualquier cosa que puedas describir en términos de **atributos y acciones**.

Un **objeto** no es más que la representación de cualquier entidad concreta o abstracta que puedas percibir o imaginar y que pueda resultar de utilidad para modelar los elementos el entorno del problema que deseas resolver.



### Autoevaluación

Tenemos un objeto bombilla, de marca ACME, que se puede encender o apagar, que

tiene una potencia de 50 vatios y ha costado 3 euros. La bombilla se encuentra en este momento apagada. A partir de esta información, ¿sabrías decir qué atributos y qué acciones (comportamiento) podríamos relacionar con ese objeto bombilla?

- Ⓐ Objeto bombilla con atributos potencia (50 vatios), precio (3 euros), marca (ACME) y estado (apagada). Las acciones que se podrían ejercer sobre el objeto serían encender y apagar.
- Ⓑ Objeto bombilla con atributos precio (3 euros), marca (ACME) y apagado. Las acciones que se podrían ejercer sobre el objeto serían encender y apagar.
- Ⓒ Objeto bombilla con atributos precio (3 euros), marca (ACME), potencia (50 vatios) y estado (apagada). No se puede ejercer ninguna acción sobre el objeto.
- Ⓓ Se trata de un objeto bombilla cuyas posibles acciones son encender, apagar y arreglar. Sus atributos serían los mismos que en el caso a).

## 4.- Clases.

Hasta ahora hemos visto lo que son los objetos. Un programa informático se compone de muchos objetos, algunos de los cuales comparten la misma estructura y comportamiento. Si tuviéramos que definir su estructura y comportamiento objeto cada vez que queremos crear un objeto, estaríamos utilizando mucho código redundante. Por ello lo que se hace es crear una **clase**, que es una descripción de un conjunto de objetos que comparten una estructura y un comportamiento común. Y a partir de la clase, se crean tantas "copias" o "instancias" como necesitemos. Esas copias son los objetos de la clase.

Las **clases** constan de **datos** y **métodos** que resumen las características comunes de un conjunto de objetos. Un programa informático está compuesto por un conjunto de clases, a partir de las cuales se crean **objetos** que interactúan entre sí.

En otras palabras, una **clase** es una **plantilla** o **prototipo** donde se especifican:

- ✓ Los **atributos** comunes a todos los objetos de la clase.
- ✓ Los **métodos** que pueden utilizarse para manejar esos objetos.

Si nos volvemos a fijar en los ejemplos de objetos del apartado anterior podríamos observar que las clases serían lo que clasificamos como "familias" de objetos (coches, cocodrilos y círculos).

En el lenguaje cotidiano de muchos programadores puede ser habitual la confusión entre los términos clase y objeto. Aunque normalmente el contexto nos permite distinguir si nos estamos refiriendo realmente a una clase (definición abstracta) o a un objeto (instancia concreta), hay que tener cuidado con su uso para no dar lugar a interpretaciones erróneas, especialmente durante el proceso de aprendizaje.

Para declarar una clase en Java se utiliza la palabra reservada **class**. La declaración de una clase está compuesta por:

- ✓ **Cabecera de la clase.** La cabecera es un poco más compleja que como aquí definimos, pero por ahora sólo nos interesa saber que está compuesta por una serie de modificadores, en este caso hemos puesto **public** que indica que es una clase pública a la que pueden acceder otras clases del programa, la palabra reservada **class** y el nombre de la clase.
- ✓ **Cuerpo de la clase.** En él se especifican encerrados entre llaves los atributos y los métodos que va a tener la clase.

```
/*
 * Estructura de una clase en Java
 */

public class NombreClase {

    // Declaracion de los atributos

    // Declaracion de los metodos

    public static void main(String[] args) {

        // Declaracion de variables y/o constantes
        // Instrucciones del metodo main
    }
}
```

En la unidad anterior ya hemos utilizado clases, aunque aún no sabíamos su significado exacto. Disponían, entre otros, del método **main()**.

El método `main()` se utiliza para indicar que se trata de una clase principal, a partir de la cual va a empezar la ejecución del programa. Este método no aparece si la clase que estamos creando no va a ser la clase principal del programa.



## Autoevaluación

**Un objeto y una clase en realidad hacen referencia al mismo concepto. Podría decirse que son sinónimos. ¿Verdadero o falso?**

- Verdadero
- Falso

## 5.- Utilización de objetos.

Una vez que hemos creado una clase, podemos crear objetos en nuestro programa a partir de esas clases.

Cuando creamos un objeto a partir de una clase se dice que hemos creado una "**instancia de la clase**". A efectos prácticos, "objeto" e "instancia de clase" son términos similares. Es decir, nos referimos a objetos como instancias cuando queremos hacer hincapié que son de una clase particular.

Los objetos se crean a partir de las clases, y representan casos individuales de éstas.

Para entender mejor el concepto entre un objeto y su clase, piensa en un **molde de galletas** y las **galletas**. El molde sería la clase, que define las características del objeto, por ejemplo su forma y tamaño. Las galletas creadas a partir de ese molde son los objetos o instancias.



Otro ejemplo, imagina una clase **Persona** que reúna las características comunes de las personas (color de pelo, ojos, peso, altura, etc.) y las acciones que pueden realizar (crecer, dormir, comer, etc.). Posteriormente dentro del programa podremos crear un objeto **Trabajador** que esté basado en esa clase **Persona**. Entonces se dice que el objeto **Trabajador** es una instancia de la clase **Persona**, o que la clase **Persona** es una abstracción del objeto **Trabajador**.

Cualquier objeto instanciado de una clase contiene una copia de todos los atributos definidos en la clase. En otras palabras, lo que estamos haciendo es reservar un espacio en la memoria del ordenador para guardar sus atributos y métodos. Por tanto, cada objeto tiene una **zona de almacenamiento propia** donde se guarda toda su información, que será distinta a la de cualquier otro objeto. A las variables miembro instanciadas también se les llama **variables instancia**. De igual forma, a los métodos que manipulan esas variables se les llama **métodos instancia**.

En el ejemplo del objeto **Trabajador**, las variables instancia serían **color\_de\_pelo**, **peso**, **altura**, etc. Y los métodos instancia serían **crecer()**, **dormir()**, **comer()**, etc.



### Autoevaluación

Las variables instancia son un tipo de variables miembro.

- Verdadero  Falso

## 5.1.- Ciclo de vida de los objetos.

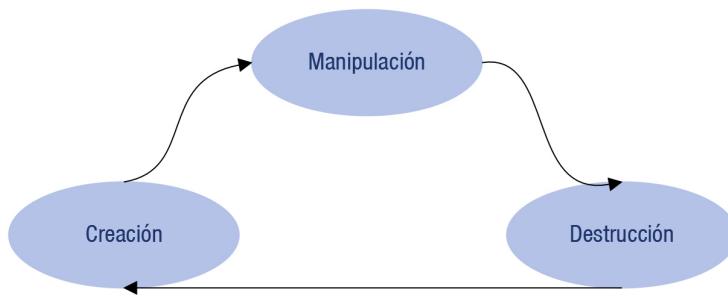
Todo programa en Java parte de una única clase, que como hemos comentado se trata de la clase principal. Esta clase ejecutará el contenido de su método **main()**, el cual será el que utilice las demás clases del programa, cree objetos y lance mensajes a otros objetos.

Las instancias u objetos tienen un tiempo de vida determinado. Cuando un objeto no se va a utilizar más en el programa, es destruido por el **recolector de basura** para liberar recursos que pueden ser reutilizados por otros objetos.

A la vista de lo anterior, podemos concluir que los objetos tienen un ciclo de vida, en el cual podemos distinguir las siguientes fases:

- ✓ **Creación**, donde se hace la reserva de memoria e inicialización de atributos.
- ✓ **Manipulación**, que se lleva a cabo cuando se hace uso de los atributos y métodos del objeto.
- ✓ **Destrucción**, eliminación del objeto y liberación de recursos.

Ciclo de vida de un objeto



### Autoevaluación

Las fases del ciclo de vida de un objeto son: Creación, Manipulación y Destrucción.

- Verdadero  Falso

## 5.2.- Destrucción de objetos y liberación de memoria.

---



Cuando un objeto deja de ser utilizado, es necesario liberar el espacio de memoria y otros recursos que poseía para poder ser reutilizados por el programa. A esta acción se le denomina **destrucción del objeto**.

En Java la destrucción de objetos corre a cargo del **recolector de basura (garbage collector)**. Es un sistema de destrucción automática de objetos que ya no son utilizados. Esto evita que los programadores tengan que preocuparse de realizar la liberación de memoria.

El recolector de basura se ejecuta en modo segundo plano y de manera muy eficiente para no afectar a la velocidad del programa que se está ejecutando. Lo que hace es que periódicamente va buscando objetos que ya no son referenciados, y cuando encuentra alguno lo marca para ser eliminado. Despues los elimina en el momento que considera oportuno.

## 6.- Utilización de métodos.

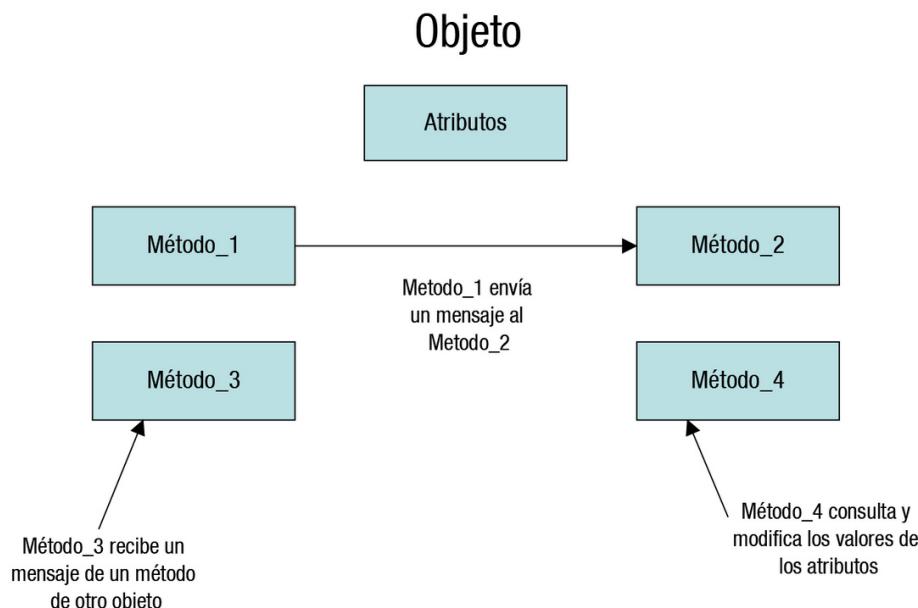
Los métodos, junto con los atributos, forman parte de la estructura interna de un objeto. Los métodos contienen la declaración de variables locales y las operaciones que se pueden realizar para el objeto, y que son ejecutadas cuando el método es invocado. Se definen en el cuerpo de la clase y posteriormente son instanciados para convertirse en **métodos instancia** de un objeto.

Para utilizar los métodos adecuadamente es conveniente conocer la estructura básica de que disponen.

Al igual que las clases, los métodos están compuestos por una **cabecera** y un **cuerpo**. La cabecera también tiene modificadores, en este caso hemos utilizado **public** para indicar que el método es público, lo cual quiere decir que le pueden enviar mensajes no sólo los métodos del objeto sino los métodos de cualquier otro objeto externo.

Dentro de un método nos encontramos el cuerpo del método que contiene el código de la acción a realizar. Las acciones que un método puede realizar son:

- ✓ **Iniciar**izar los atributos del objeto
- ✓ **Consultar** los valores de los atributos
- ✓ **Modificar** los valores de los atributos
- ✓ **Llamar** a otros métodos, del mismo del objeto o de objetos externos

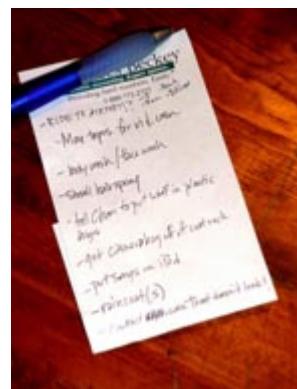


## 6.1.- Parámetros y valores devueltos.

Los métodos se pueden utilizar tanto para consultar información sobre el objeto como para modificar su estado. La información consultada del objeto se devuelve a través de lo que se conoce como **valor de retorno**, y la modificación del estado del objeto, o sea, de sus atributos, se hace mediante la **lista de parámetros**.

En general, la lista de parámetros de un método se puede declarar de dos formas diferentes:

- ✓ **Por valor.** El valor de los parámetros no se devuelve al finalizar el método, es decir, cualquier modificación que se haga en los parámetros no tendrá efecto una vez se salga del método. Esto es así porque cuando se llama al método desde cualquier parte del programa, dicho método recibe una copia de los argumentos, por tanto cualquier modificación que haga será sobre la copia, no sobre las variables originales.
- ✓ **Por referencia.** La modificación en los valores de los parámetros sí tienen efecto tras la finalización del método. Cuando pasamos una variable a un método por referencia lo que estamos haciendo es pasar la dirección del dato en memoria, por tanto cualquier cambio en el dato seguirá modificado una vez que salgamos del método.



En el lenguaje Java, todas las variables se pasan por valor, excepto los objetos que se pasan por referencia. En Java, la declaración de un método tiene dos restricciones:

- ✓ **Un método siempre tiene que devolver un valor** (no hay valor por defecto). Este **valor de retorno** es el valor que devuelve el método cuando termina de ejecutarse, al método o programa que lo llamó. Puede ser un tipo primitivo, un tipo referenciado o bien el tipo void, que indica que el método no devuelve ningún valor.
- ✓ **Un método tiene un número fijo de argumentos.** Los argumentos son variables a través de las cuales se pasa información al método desde el lugar del que se llame, para que éste pueda utilizar dichos valores durante su ejecución. Los argumentos reciben el nombre de **parámetros** cuando aparecen en la declaración del método.

El **valor de retorno** es la información que devuelve un método tras su ejecución.

Según hemos visto en el apartado anterior, la cabecera de un método se declara como sigue:

```
public tipo_de_dato_devuelto nombre_metodo (lista_de_parametros);
```

Como vemos, el tipo de dato devuelto aparece después del modificador public y se corresponde con el valor de retorno.

La lista de parámetros aparece al final de la cabecera del método, justo después del nombre, encerrados entre signos de paréntesis y separados por comas. Se debe indicar el tipo de dato de cada parámetro así:

```
(tipo_parametro1 nombre_parametro1, ..., tipo_parametroN nombre_parametroN)
```

Cuando se llame al método, se deberá utilizar el nombre del método, seguido de los argumentos que deben coincidir con la lista de parámetros.

La **lista de argumentos** en la llamada a un método debe coincidir en número, tipo y orden con

los **parámetros** del método, ya que de lo contrario se produciría un error de sintaxis.

## 6.2.- Constructores.

Son métodos especiales que sirven para inicializar valores.

**Un constructor es un método especial con el mismo nombre de la clase y que no devuelve ningún valor tras su ejecución.**

Cuando creamos un objeto debemos instanciarlo utilizando el constructor de la clase. La estructura de los constructores es similar a la de cualquier método, salvo que no tiene tipo de dato devuelto porque no devuelve ningún valor. Está formada por una cabecera y un cuerpo, que contiene la inicialización de atributos y resto de instrucciones del constructor.

El método constructor tiene las siguientes particularidades:

- ✓ El constructor **es invocado automáticamente** en la creación de un objeto, y sólo esa vez.
- ✓ Los constructores **no empiezan con minúscula**, como el resto de los métodos, ya que se llaman igual que la clase y las clases empiezan con letra mayúscula.
- ✓ **Puede** haber **varios constructores** para una clase.
- ✓ Como cualquier método, el constructor puede tener **parámetros** para definir qué valores dar a los atributos del objeto.
- ✓ El constructor **por defecto** es aquél que no tiene argumentos o parámetros. Cuando creamos un objeto llamando al nombre de la clase sin argumentos, estamos utilizando el constructor por defecto.
- ✓ Es necesario que toda clase tenga **al menos un constructor**. Si no definimos constructores para una clase, y sólo en ese caso, el compilador crea un constructor por defecto vacío, que inicializa los atributos a sus valores por defecto, según del tipo que sean: **0** para los tipos numéricos, **false** para los **boolean** y **null** para los tipo carácter y las referencias. Dicho constructor lo que hace es llamar al constructor sin argumentos de la superclase (clase de la cual hereda); si la superclase no tiene constructor sin argumentos se produce un error de compilación.

Cuando definimos constructores personalizados, el constructor por defecto deja de existir, y si no definimos nosotros un constructor sin argumentos cuando intentemos utilizar el constructor por defecto nos dará un error de compilación.