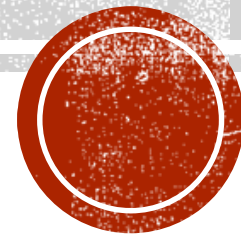


PROGRAMACIÓN ORIENTADA A OBJETOS



Ing. Sara Osorio
Especialista en Ingeniería de Software
UNINPAHU
Facultad de Ingeniería
2019

Introducción

Conceptos fundamentales

Clase

Objeto

Atributo

Método

Evento

Mensaje

Estado interno

Componentes e Identificación de un objeto

Características de la POO

Abstracción

Encapsulamiento

Principio de ocultación

Polimorfismo

Herencia

Recolección de basura



P. O. O.

La Programación Orientada a Objetos (POO) es el **paradigma de programación** más utilizado en la actualidad.



Características

- Marca
- Color
- Tipo
- Precio
- No. de Puertas
- Tipo Combustible
- Cilindros
- Transmisión

Acciones

- Encender
- Avanzar
- Retroceder
- Detener
- Apagar

Características

- Nombre
- Especie
- Color
- Edad

Acciones

- Comer
- Dormir
- Correr

La POO se refiere a transformar el mundo real en código.

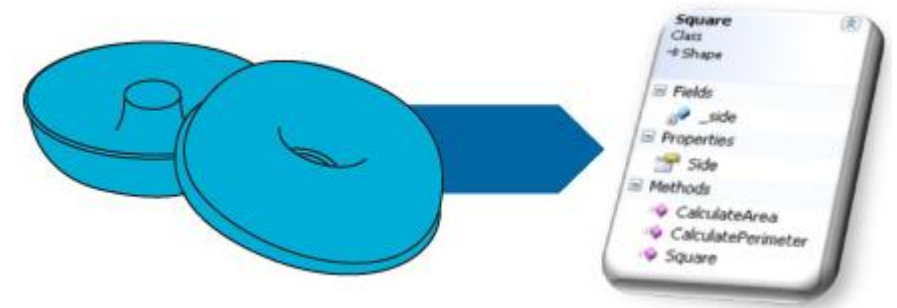
En las imágenes vemos 2 objetos: un automóvil y un león. Como objetos ambos tienen rasgos que los caracterizan (Propiedades) y pueden realizar acciones (métodos). Una vez identificados, debemos proceder a transportar este concepto a nuestro código. Generando primeramente nuestras Clases, que nos servirán para la creación de nuestros objetos.



CLASES

Así como en diseño electrónico se usan prototipos de dispositivos que podrían utilizarse en repetidas ocasiones para construir los dispositivos reales, una clase es un prototipo de software que puede utilizarse para instanciar (es decir crear) muchos objetos iguales.

También lo podemos ver como un molde, que permite crear muchos objetos con la misma forma y tamaño. Ese molde sería nuestra clase y la instancia de una clase sería nuestro objeto.



OBJETOS

Representa una instancia de un elemento del mundo del problema

Es cualquier elemento que se pueda describir en términos de su estado y su comportamiento

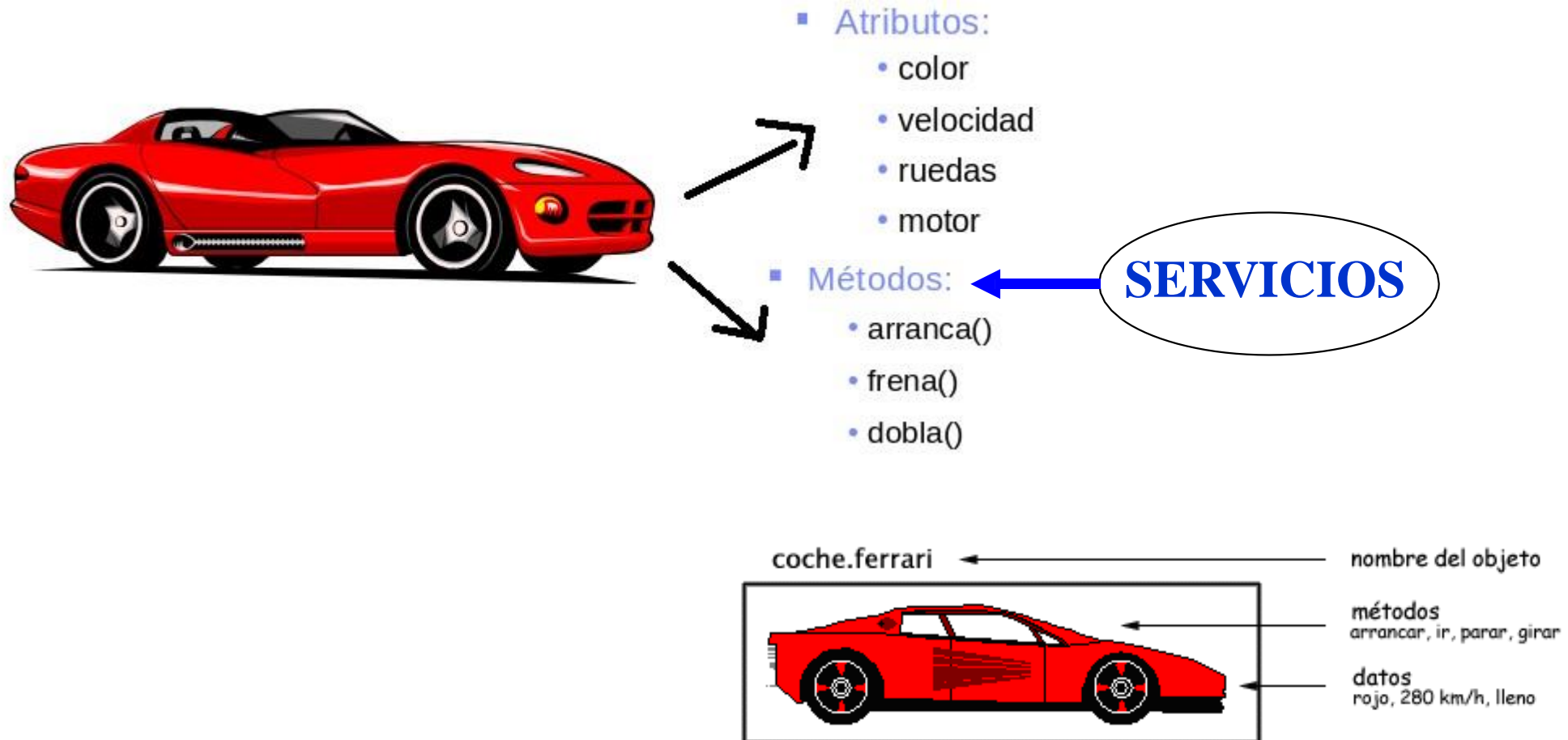


ESTADO

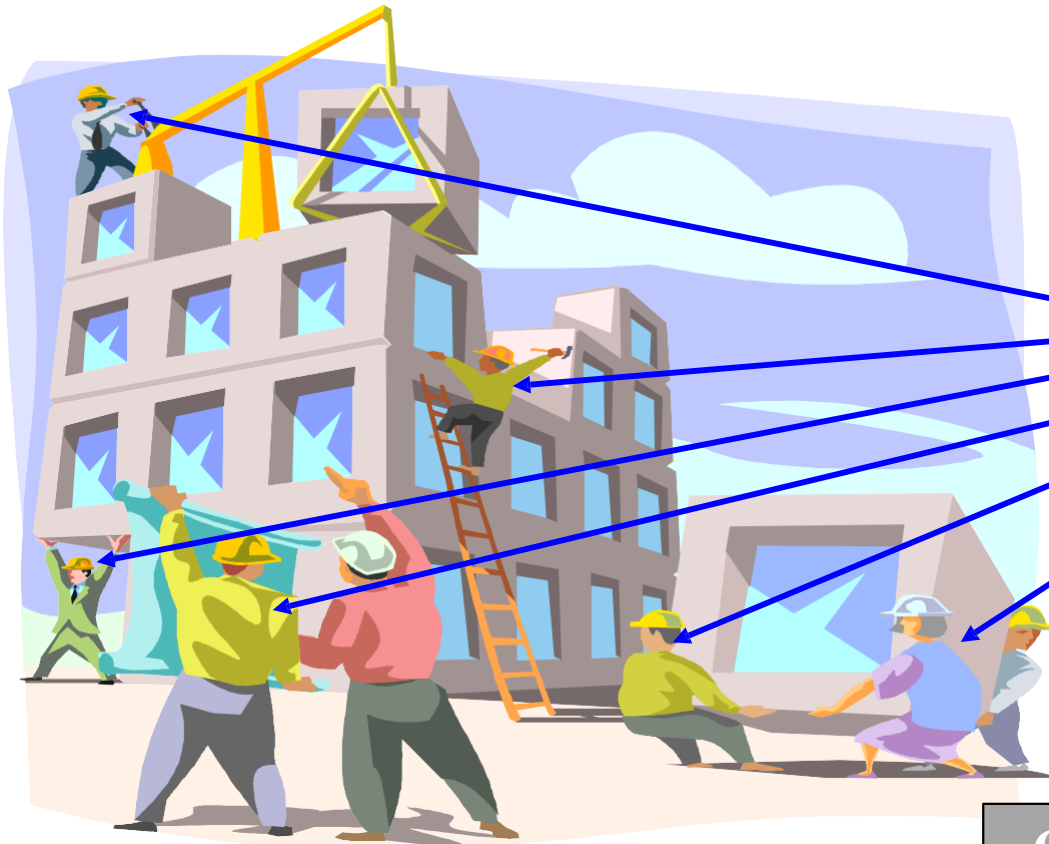
- Placa
- Capacidad
- color
- Marca
- Fabricante



CARACTERÍSTICAS DE LOS OBJETOS



COLABORACIÓN



REQUERIMIENTO
CONSTRUIR UN EDIFICIO

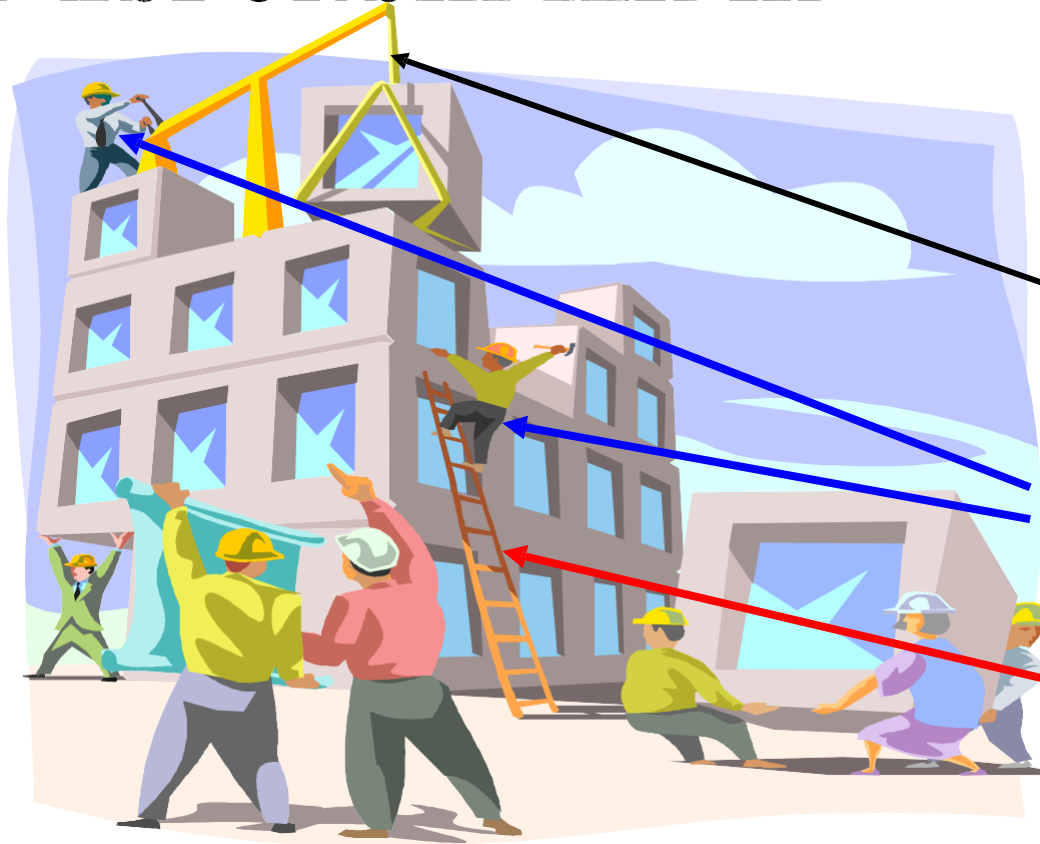
OBJETO OBRERO

CADA UNO TIENE SU
PROPIO ESTADO

CADA UNO ASUME PARTE
DE LA RESPONSABILIDAD
CON SU COMPORTAMIENTO



UNICA ESPONSABILIDAD



OBJETOS DISTINTOS

OBJETO CARGADOR

OBJETO OBRERO

OBJETO ESCALERA

Objetos con responsabilidades diferentes, interactúan entre si para cumplir objetivo.



CLASES

Agrupar un conjunto de objetos del mundo del problema que tienen **las mismas características y el mismo comportamiento**

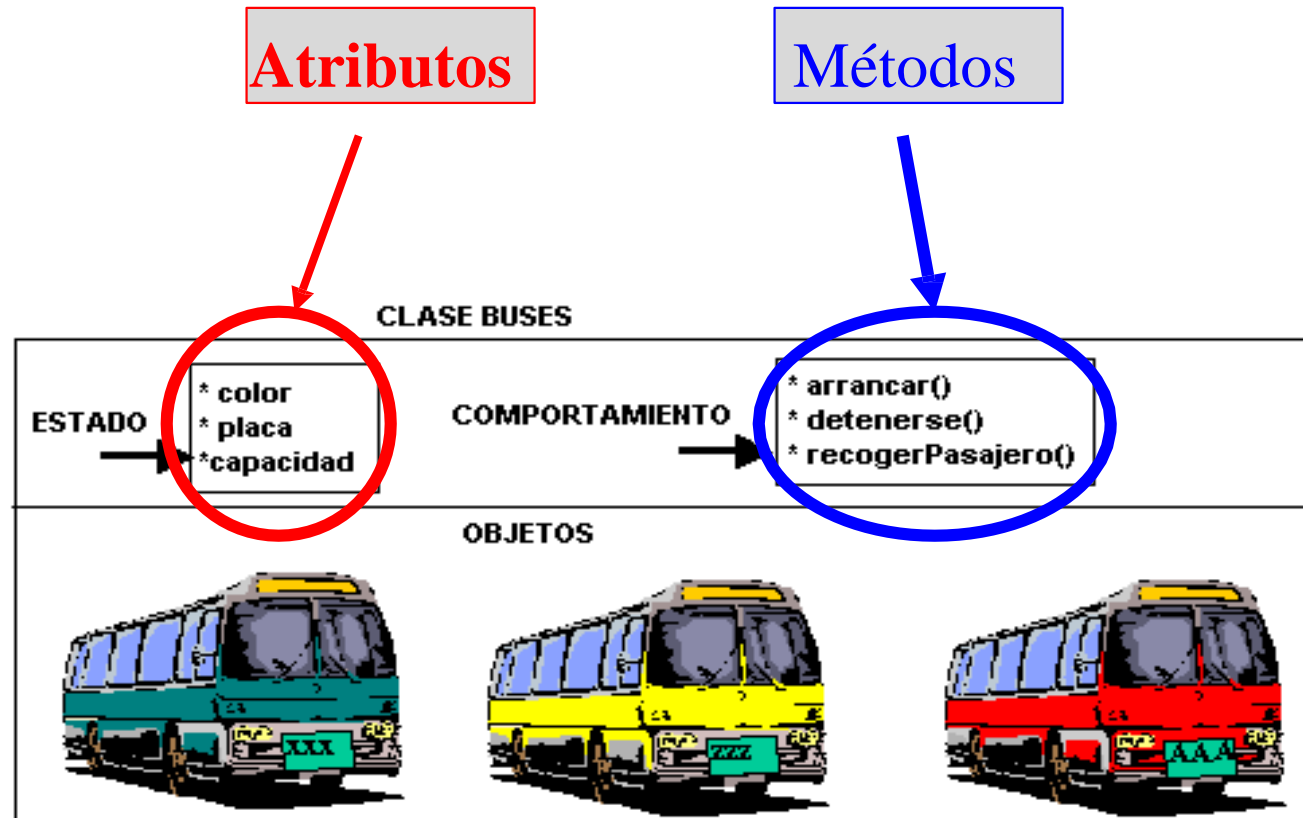
Son plantillas para crear objetos

Un objeto es una instancia de una clase

Ej. Receta para hacer galletas de chocolate. Con ella creas una o muchas galletas de chocolate



ATRIBUTOS Y MÉTODOS



ATRIBUTOS

Los atributos son las características individuales que diferencian un objeto de otro y determinan su apariencia, estado u otras cualidades. Los atributos se guardan en variables denominadas de instancia, y cada objeto particular puede tener valores distintos para estas variables.

Las variables de instancia también denominados miembros dato, son declaradas en la clase pero sus valores son fijados y cambiados en el objeto.

Además de las variables de instancia hay variables de clase, las cuales se aplican a la clase y a todas sus instancias. Por ejemplo, el número de ruedas de un automóvil es el mismo cuatro, para todos los automóviles.



ATRIBUTOS

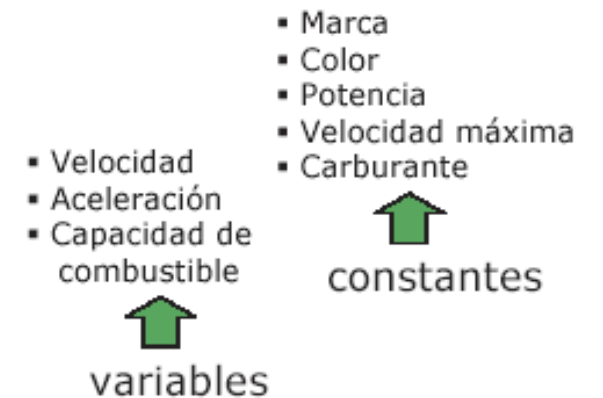
Describen el estado de un objeto

Objetos simples

- Tipos de datos primitivos
Ej. int, float, double, char, etc

Objetos Complejos

- Referencias a otros objetos
Ej. Estructuras de datos u otros objetos



MÉTODOS

Consiste en la implementación en una clase de un protocolo de respuesta a los mensajes dirigidos a los objetos de la misma. La respuesta a tales mensajes puede incluir el envío por el método de mensajes al propio objeto y aun a otros, también como el cambio del estado interno del objeto

Un objeto puede realizar una serie de acciones o puede ofrecer una serie de servicios.



- Arrancar motor
- Parar motor
- Acelerar
- Frenar
- Girar a la derecha (grados)
- Girar a la izquierda (grados)
- Cambiar marcha (nueva marcha)



método



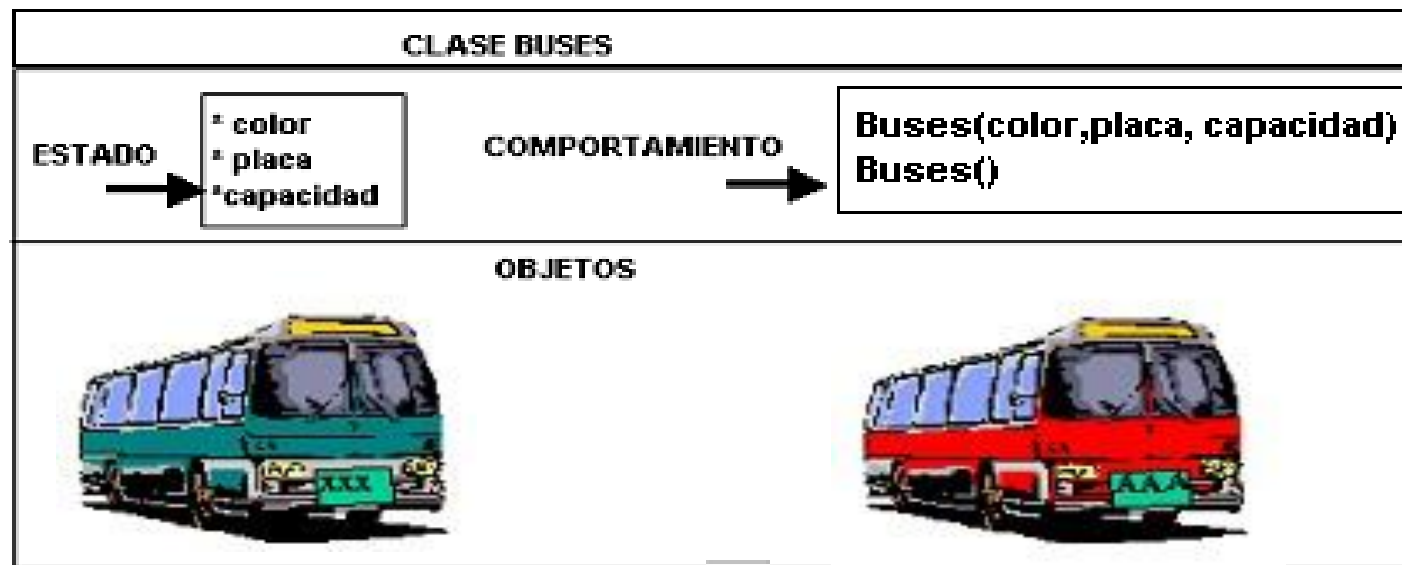
argumentos



TIPOS DE MÉTODOS

Constructores

Son fragmentos de código que sirven para inicializar un **objeto** a un estado determinado.



`Buses(verde, xxx, 20)`

`Buses(Rojo, AAA, 20)`



TIPOS DE MÉTODOS

Analizadores

Son fragmentos de código que permiten obtener el estado del objeto



Objeto Conductor

color = getColor()

COMPORTAMIENTO
getColor():Color
getCapacidad():int
getPlaca():Placa

* color : VERDE
* placa : XXX
* capacidad : 50



TIPOS DE MÉTODOS

Modificadores

Son fragmentos de código que permiten cambiar el estado del objeto



Objeto Conductor

setColor(amarillo)

COMPORTAMIENTO
setColor(Color)
setCapacidad(capacid)
setPlaca(Placa)

* color	AMARILLO
* placa	: XXX
* capacidad	: 50



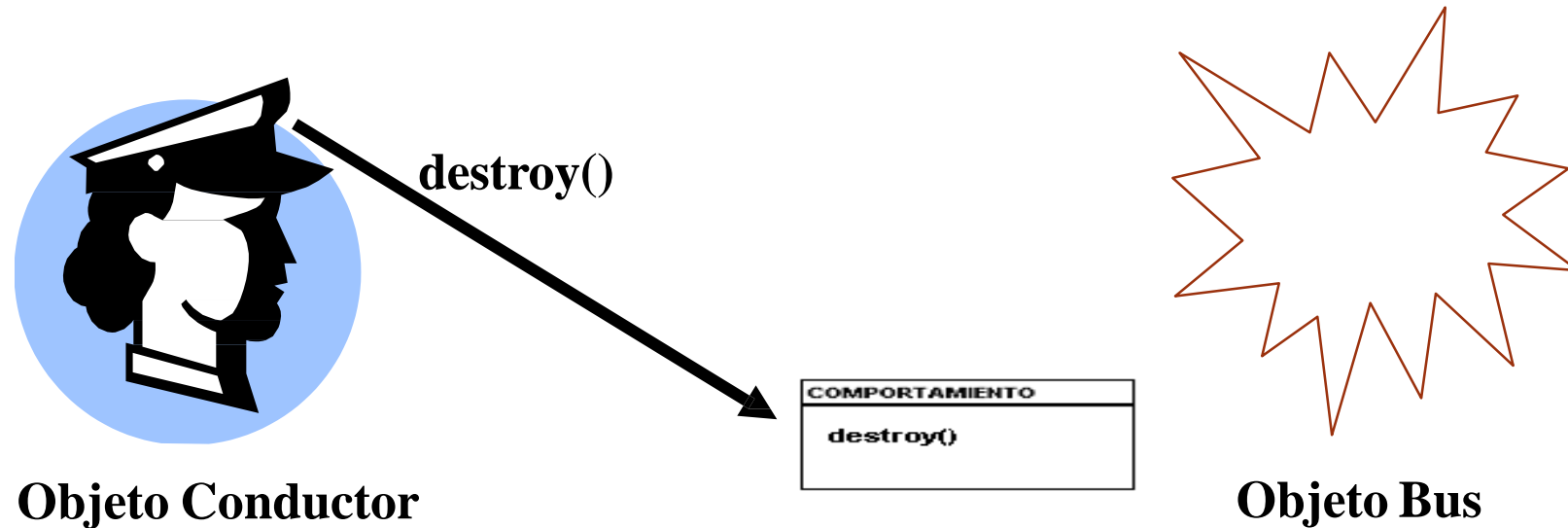
Objeto Bus



TIPOS DE MÉTODOS

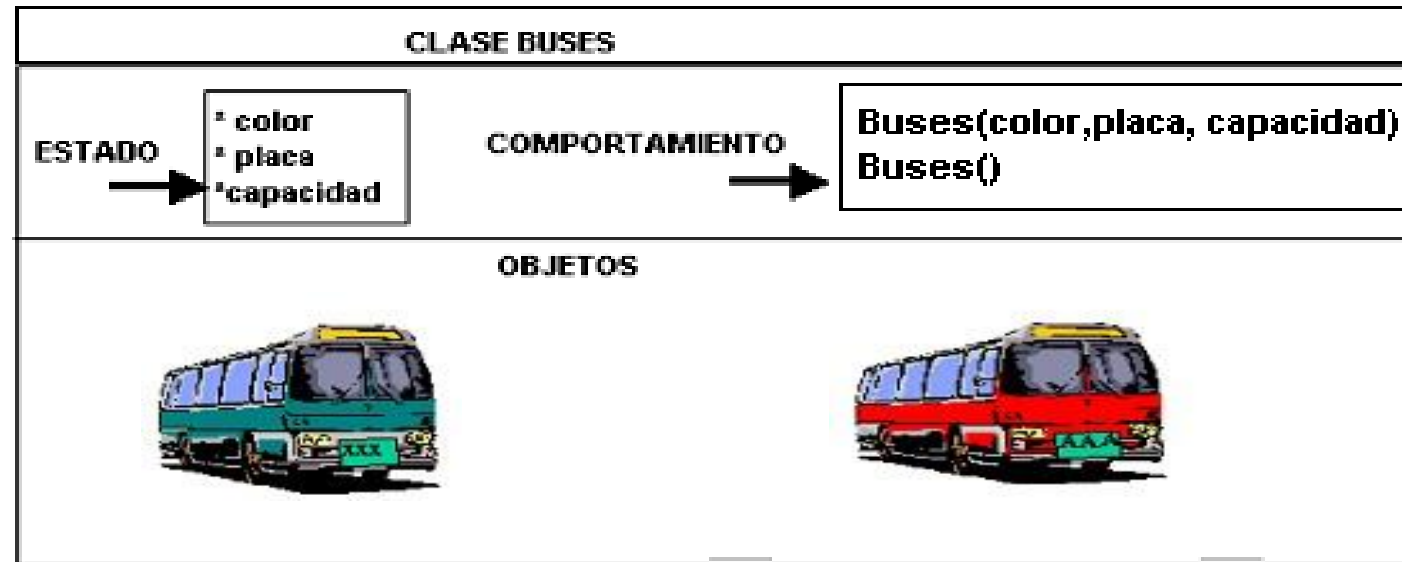
Destructores

Son fragmentos de código que permiten eliminar un objeto



SOBRECARGA DE MÉTODOS

Métodos dentro de una misma clase que tienen el mismo nombre pero distinta firma



Buses() /* valores por defecto : color = Rojo, Placa =AAA, cap=20 */

Buses(verde, xxx, 20)



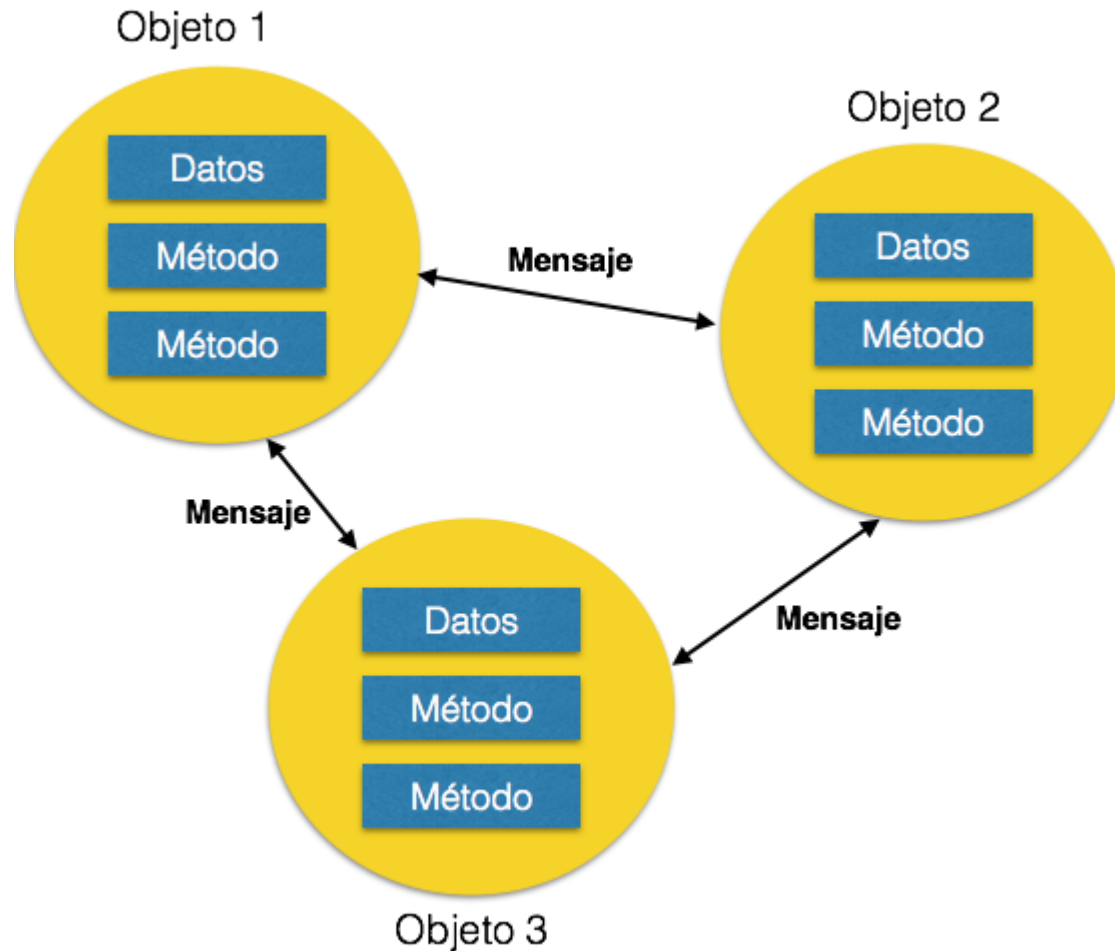
EVENTO

Es un suceso en el sistema (tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto).

El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente. También se puede definir como evento, a la reacción que puede desencadenar un objeto, es decir la acción que genera..



MENSAJE



Una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros asociados al evento que lo generó.

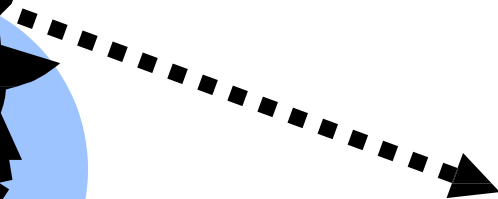


MENSAJE

Mecanismo de comunicación ente objetos para solicitar servicios



Objeto Conductor



COMPORTAMIENTO
Buses(color,placa, capacidad)
Buses()
getColor():Color
getPlaca():Placa
getCapacidad():int
setColor(color)
setPlaca(placa)
setCapacidad(capacidad)
destroy()

* color	: VERDE
* placa	: XXX
*capacidad	: 50



Objeto Bus



ESTADO INTERNO

Es una variable que se declara privada, que puede ser únicamente accedida y alterada por un método del objeto, y que se utiliza para indicar distintas situaciones posibles para el objeto (o clase de objetos).

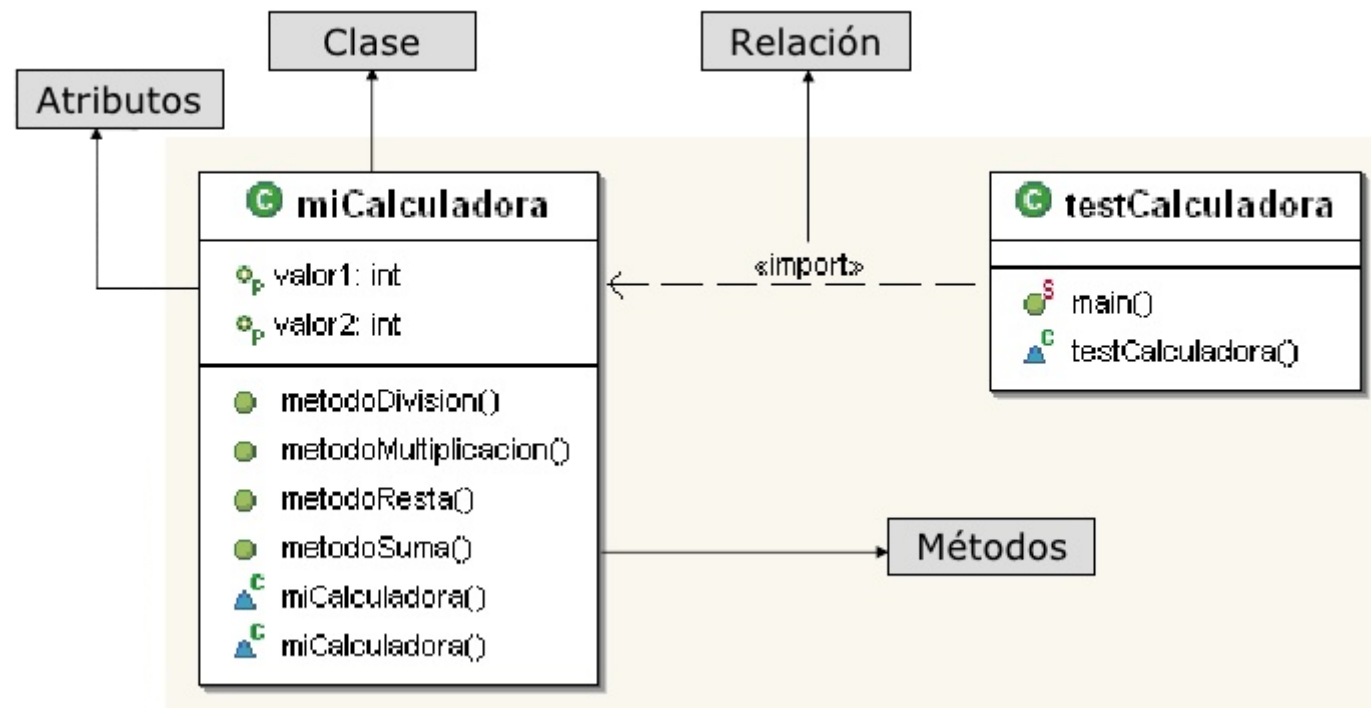
No es visible al programador que maneja una instancia de la clase.



COMPONENTES E IDENTIFICACIÓN DE UN OBJETO

Componentes de un objeto

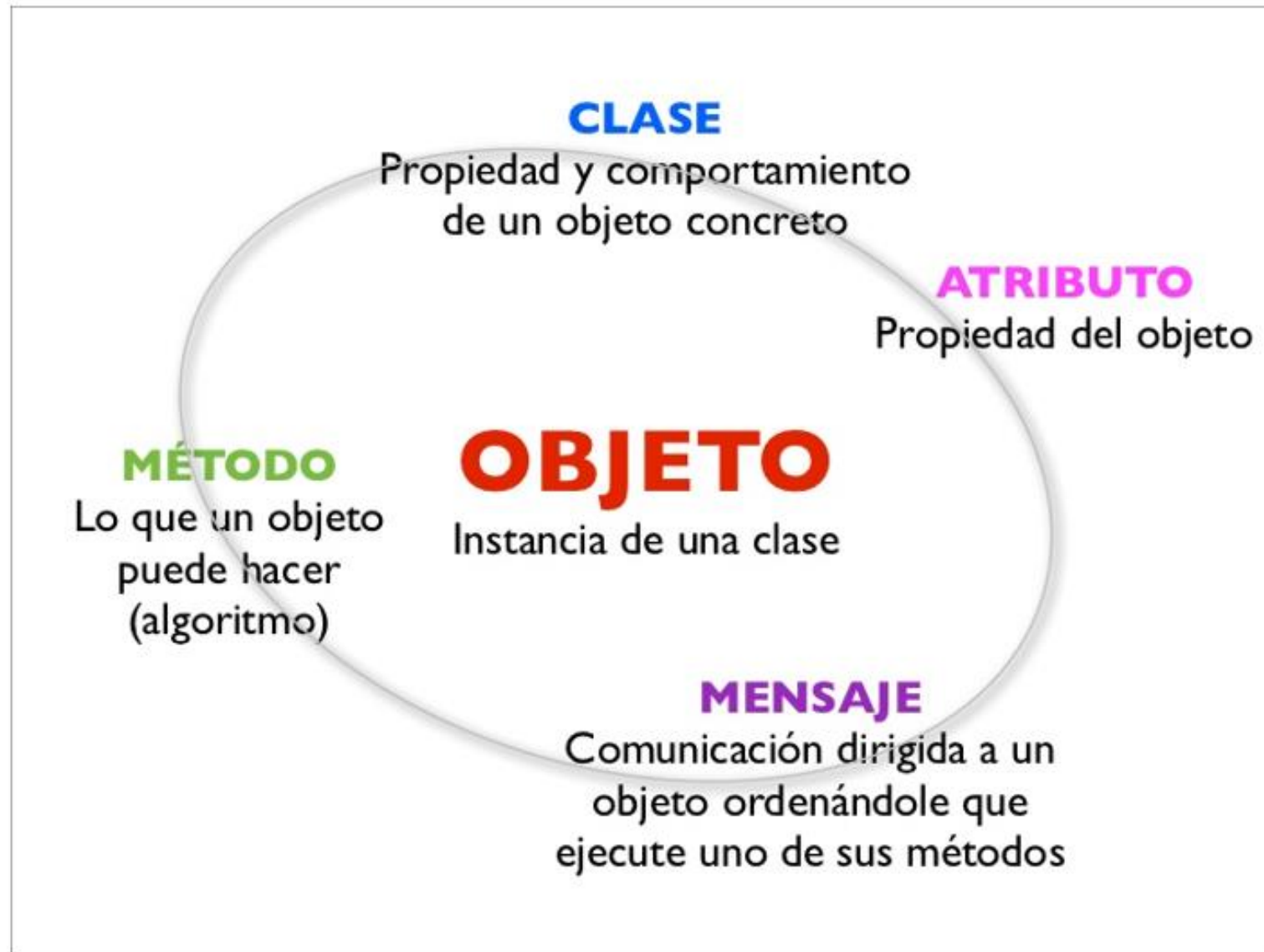
Atributos, relaciones y métodos.



Identificación de un objeto

Un objeto se representa por medio de una tabla o entidad que esté compuesta por sus atributos y métodos correspondientes.





ABSTRACCIÓN

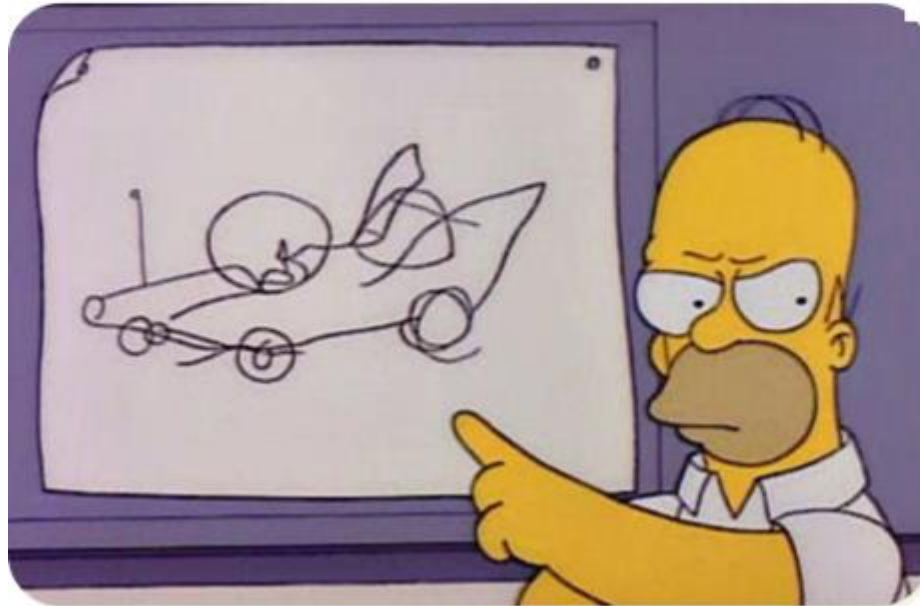
Denota las características esenciales de un objeto, donde se capturan sus comportamientos. Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar cómo se implementan estas características.

El proceso de abstracción permite seleccionar las características relevantes dentro de un conjunto e identificar comportamientos comunes para definir nuevos tipos de entidades en el mundo real.

La abstracción es clave en el proceso de análisis y diseño orientado a objetos, ya que mediante ella podemos llegar a armar un conjunto de clases que permitan modelar la realidad o el problema que se quiere atacar.



ABSTRACCIÓN



Homero Simpson construyendo el auto de sus sueños

Énfasis en el
¿qué hace? mas
que en el ¿cómo
lo hace?



ENCAPSULAMIENTO

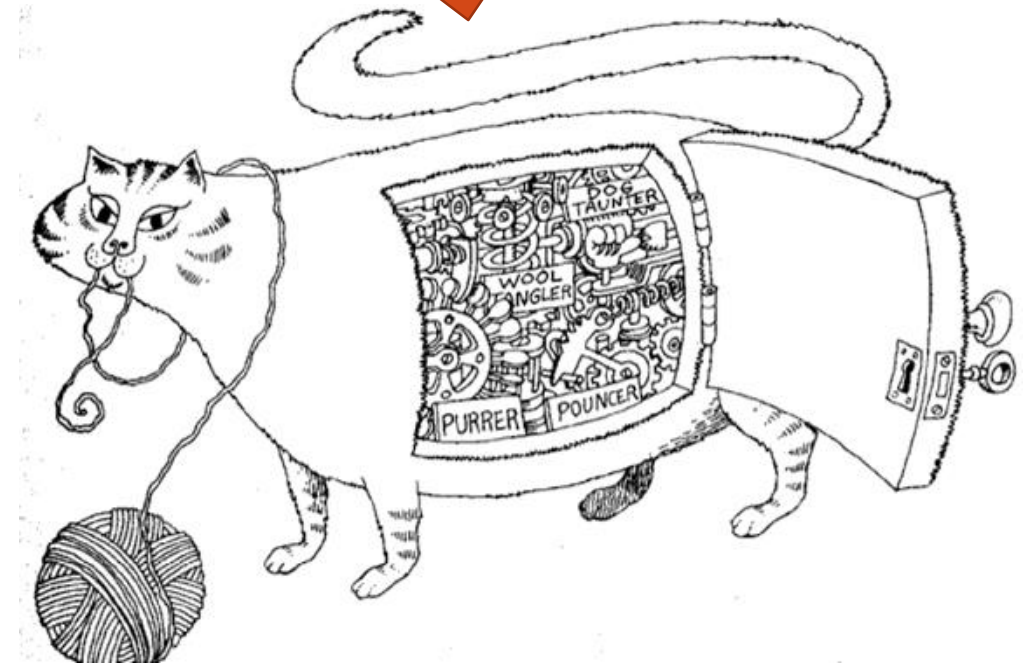
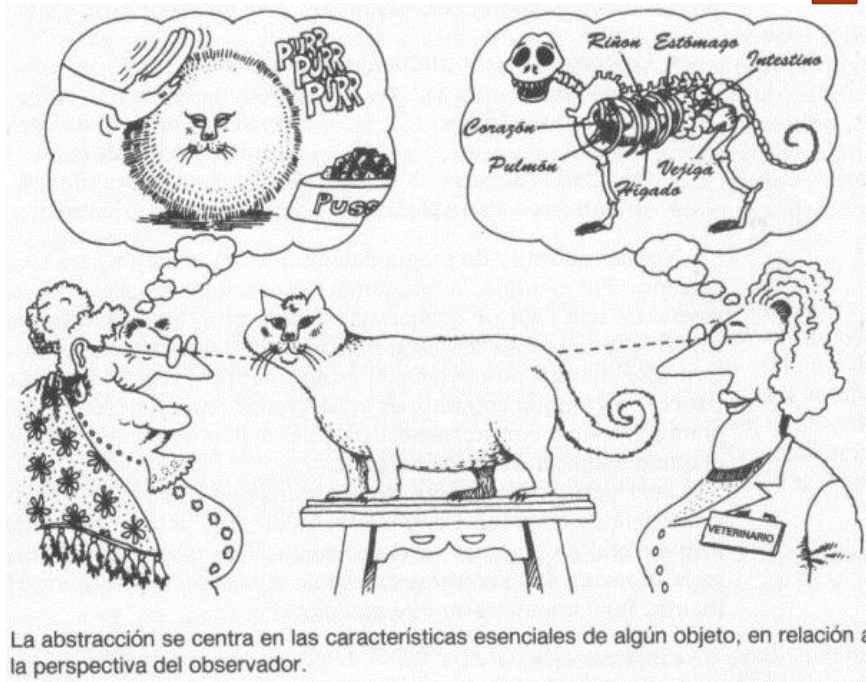
Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción.

Esto permite aumentar la cohesión de los componentes del sistema.

Algunos autores confunden este concepto con el principio de ocultación, principalmente porque se suelen emplear conjuntamente.



ENCAPSULAMIENTO



PRINCIPIO DE OCULTACIÓN

Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una interfaz a otros objetos que especifica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas, solamente los propios métodos internos del objeto pueden acceder a su estado.

Esto asegura que otros objetos no pueden cambiar el estado interno de un objeto de maneras inesperadas, eliminando efectos secundarios e interacciones inesperadas. Algunos lenguajes relajan esto, permitiendo un acceso directo a los datos internos del objeto de una manera controlada y limitando el grado de abstracción. La aplicación entera se reduce a un agregado o rompecabezas de objetos.



PRINCIPIO DE OCULTACIÓN

Visibilidad Pública: Los otros objetos pueden referenciar directamente el atributo

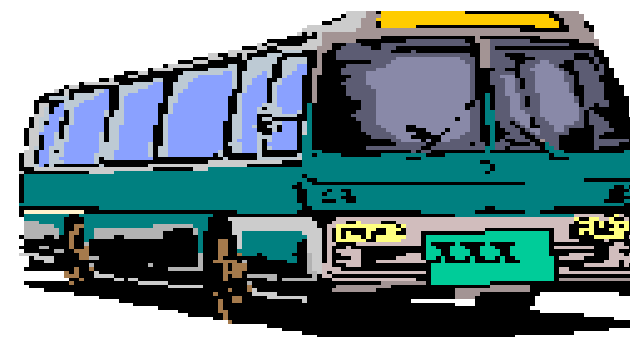


Objeto Conductor

Color = VERDE



```
* color      : VERDE  
* placa     : XXX  
* capacidad : 50
```

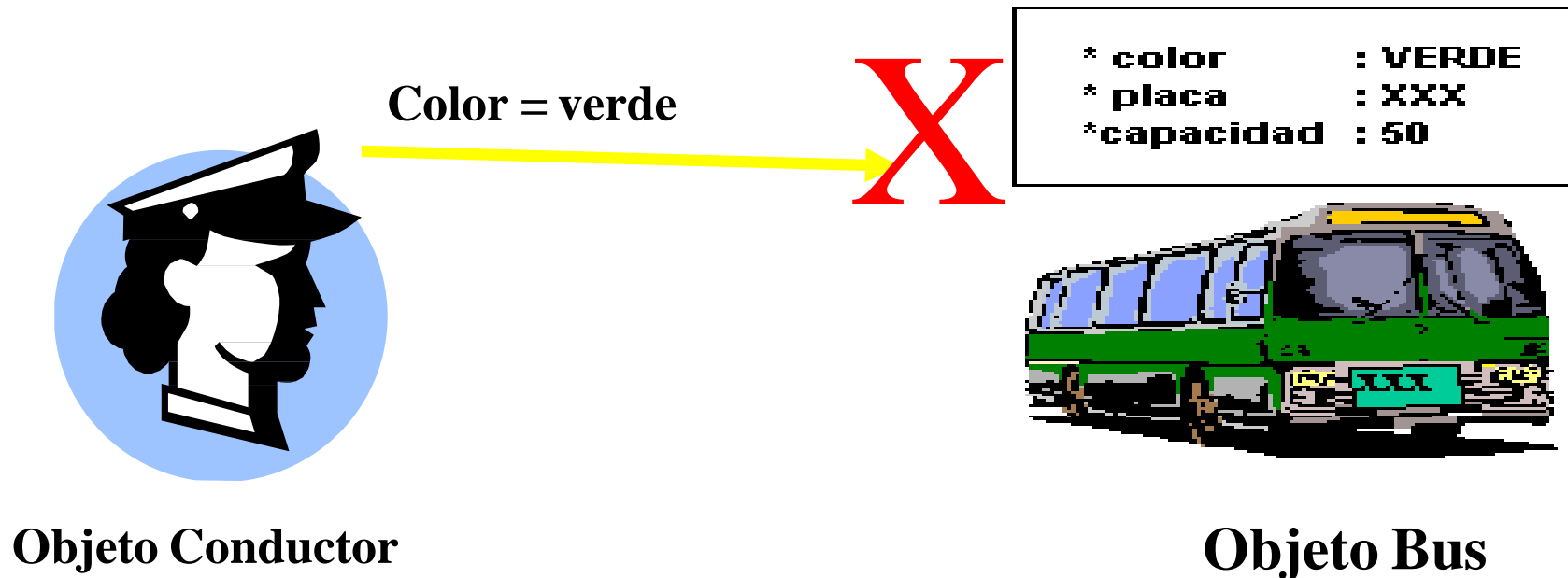


Objeto Bus



PRINCIPIO DE OCULTACIÓN

Visibilidad Privada : solo el objeto puede referenciar directamente sus atributos



PRINCIPIO DE OCULTACIÓN

Visibilidad Protegido : Es un punto medio entre público y privado, porque -como ocurre con las privadas- no se puede acceder a ella desde una instancia de la clase, pero -como ocurre con las públicas- puede ser accedido desde las subclases de ésta, no importa si se encuentran o no en el mismo paquete. Básicamente significa que, si una clase hereda de otra, tendrá acceso a las variables/funciones protegidas de la super-clase, de lo contrario, no podrá acceder a ellas.



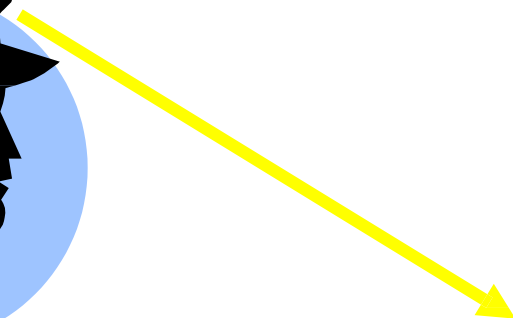
PRINCIPIO DE OCULTACIÓN

Todos los atributos deben tener visibilidad privada

El resto de objetos no conocen los detalles de la implementación



Objeto conductor



Comportamiento
setColor(Color)
getColor() : Color
.....













* color	: AMARILLO
* placa	: XXX
* capacidad	: 50



Objeto Bus



PRINCIPIO DE OCULTACIÓN

Visibilidad	Public	Private	Protected	Default*
Desde la misma clase				
Desde una subclase				
Desde otra clase (no subclase)				

**Con default, me refiero a omitir el modificador de acceso.*

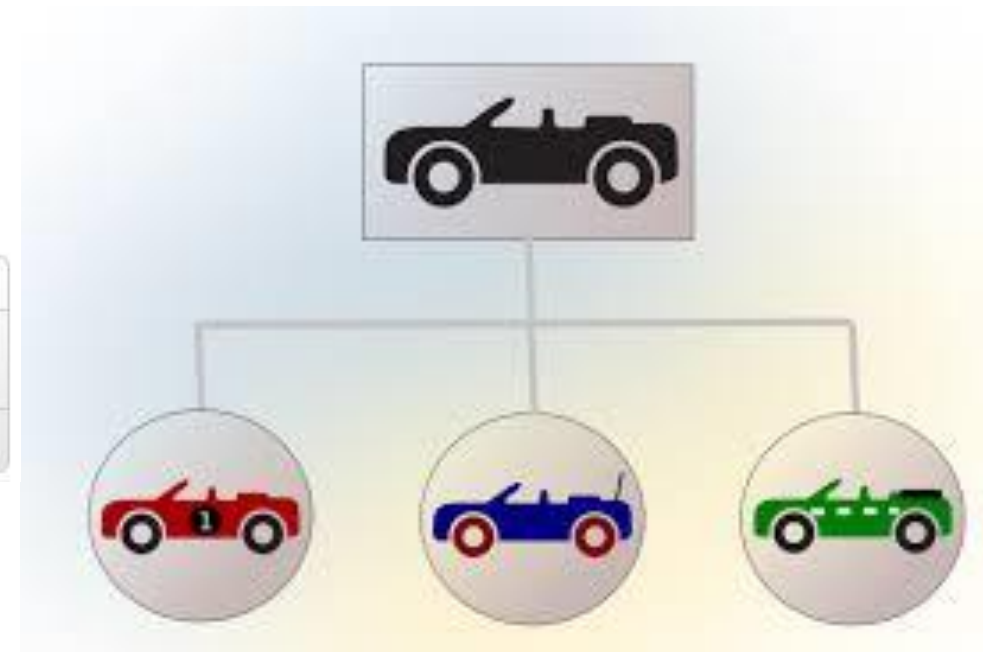
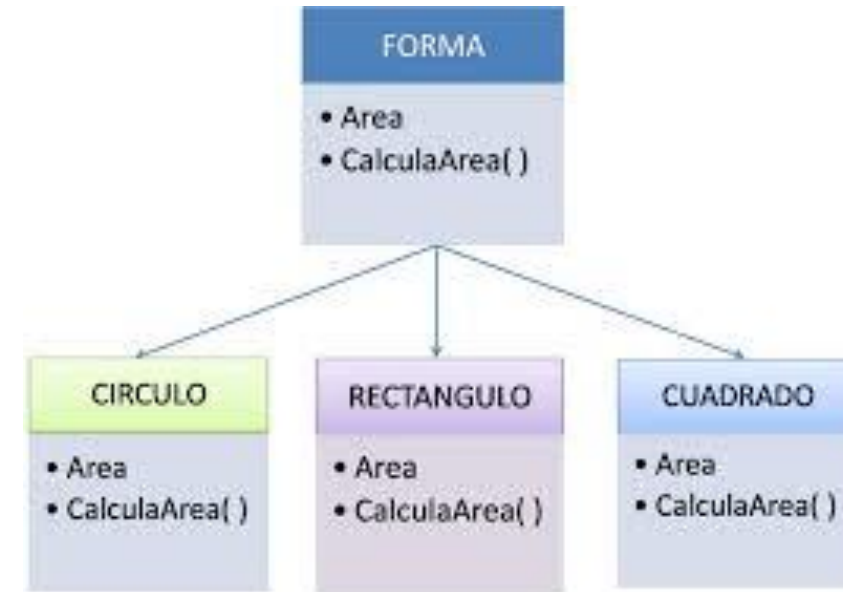
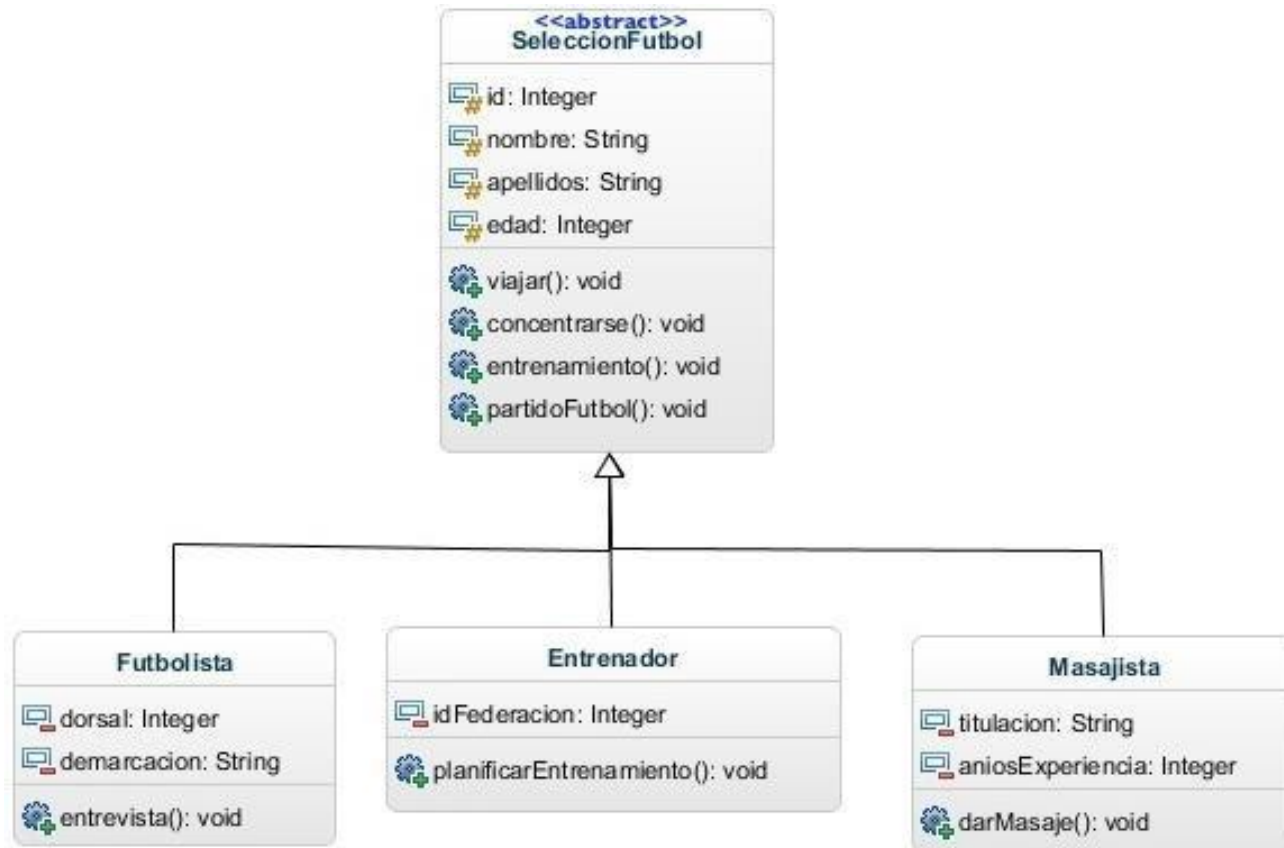


POLIMORFISMO

Comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando. O dicho de otro modo, las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del objeto referenciado. Cuando esto ocurre en "tiempo de ejecución", esta última característica se llama asignación tardía o asignación dinámica. Algunos lenguajes proporcionan medios más estáticos (en "tiempo de compilación") de polimorfismo, tales como las plantillas y la sobrecarga de operadores de C++.



POLIMORFISMO



HERENCIA

Las clases no están aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen.

La herencia organiza y facilita el polimorfismo y el encapsulamiento permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que volver a implementarlo. Esto suele hacerse habitualmente agrupando los objetos en clases y estas en árboles o enrejados que reflejan un comportamiento común. Cuando un objeto hereda de más de una clase se dice que hay herencia múltiple.



HERENCIA

La herencia nos permite, entre otras cosas, evitar tener que escribir el mismo código una y otra vez, puesto que al definir que una categoría (que en programación llamaremos clase) pertenece a otra, automáticamente estamos atribuyéndoles las características generales de la primera, sin tener que definirlas de nuevo.



RECOLECCIÓN DE BASURAS

La recolección de basura o garbage collector es la técnica por la cual el entorno de objetos se encarga de destruir automáticamente, y por tanto desvincular la memoria asociada, los objetos que hayan quedado sin ninguna referencia a ellos. Esto significa que el programador no debe preocuparse por la asignación o liberación de memoria, ya que el entorno la asignará al crear un nuevo objeto y la liberará cuando nadie lo esté usando.

En la mayoría de los lenguajes híbridos que se extendieron para soportar el Paradigma de Programación Orientada a Objetos como C++ u Object Pascal, esta característica no existe y la memoria debe desasignarse manualmente.

