

# **Introducción a la Programación Orientada a Objetos (POO)**

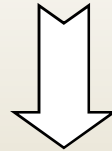
Cátedra de Programación  
de Computadoras

# Agenda

- Motivación
- Definiciones básicas
  - Objetos
  - Mensajes y métodos
  - Clases, subclases y objetos
  - Herencia
- Conceptos claves
  - Encapsulamiento
  - Abstracción
  - Polimorfismo

- Qué vimos hasta ahora?

## Programación Estructurada



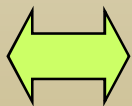
La idea principal de esta forma de programación es separar las partes complejas del programa en módulos, que sean ejecutados a medida que sean necesarios. Estos módulos son independientes entre sí, y además deben poder comunicarse.

## Problemas de la Programación Estructurada

Varios programadores trabajan en equipo desarrollando una aplicación grande.



Más de un programador manipula funciones separadas que pueden referirse a tipos de datos mutuamente compartidos, y los cambios de un programador se deben reflejar en el trabajo del resto del equipo.



¿Qué pasa si uno de los programadores decide que una estructura existente en el sistema en vez de representarse con una lista, ahora se representa con un arreglo?

Este es uno de los problemas de la programación estructurada, por lo cual se siguió investigando sobre diferentes metodologías de programación.

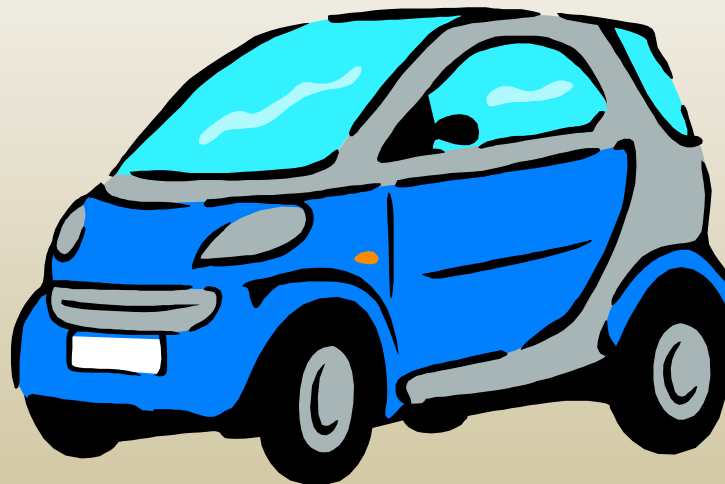
# Introducción – Programación Orientada a Objetos

Qué es lo que ves?



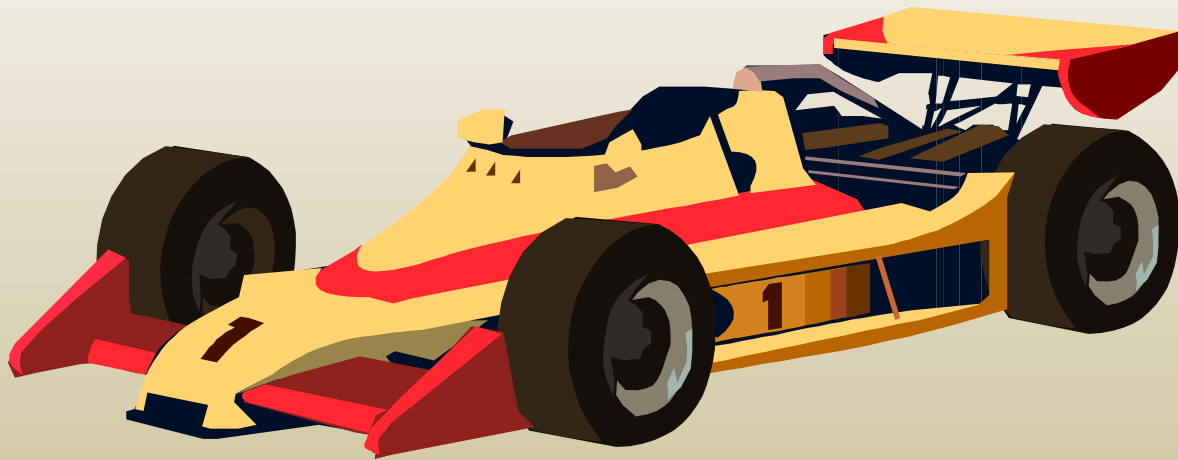
# Introducción – Programación Orientada a Objetos

Qué es lo que ves?



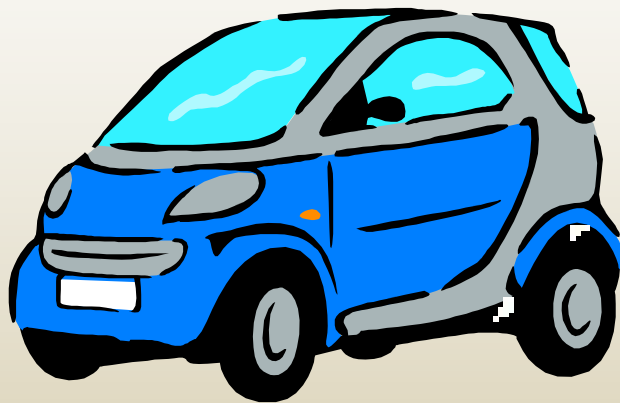
# Introducción – Programación Orientada a Objetos

Qué es lo que ves?



# Introducción – Programación Orientada a Objetos

Qué es lo que tienen en común?



Modelo  
Marca  
Color  
Velocidad

Acelerar  
Desacelerar  
Apagar  
Arrancar

Se podría encontrar una forma de definir “algo” que encapsule las características y comportamiento comunes



# Introducción – Programación Orientada a Objetos -CLASES

## Qué es una clase?

Es un modelo o prototipo que define las variables y métodos comunes a todos los objetos de ciertas características comunes.

Es una plantilla genérica para un conjunto de objetos de similares características.

Contiene:

- Conjunto de atributos comunes
- Estructura de datos
- Comportamiento por medio de métodos

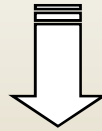
Cómo lo relacionamos  
con nuestro ejemplo  
de los autos?

Deberíamos implementar la clase auto

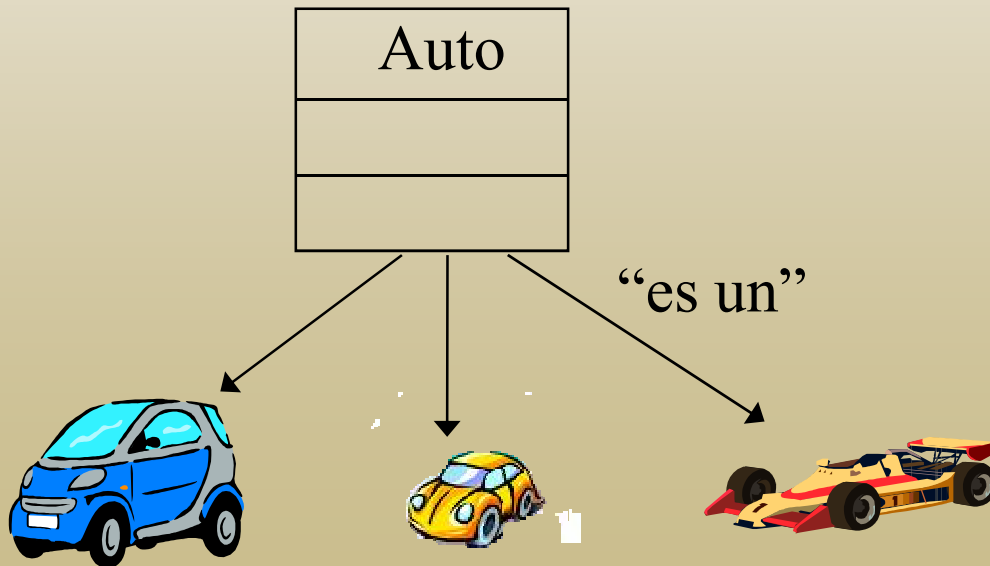


# Programación Orientada a Objetos -CLASES

Cada uno de los diferentes autos vistos anteriormente tienen características comunes pero con valores diferentes. Es decir los tres autos tienen color pero cada uno un color diferente.

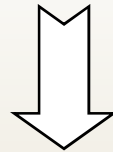


Instancia de una clase = OBJETO



# Programación Orientada a Objetos

## Programación Orientada a Objetos



Surge de la evolución de la programación estructurada y básicamente simplifica la programación con la nueva filosofía y nuevos conceptos que tiene.

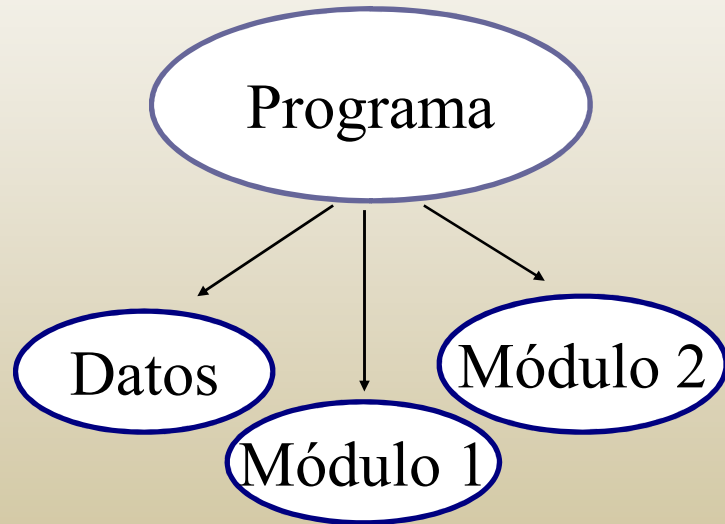
La POO se basa en dividir el sistema en componentes que contienen operaciones y datos. Cada componente se denomina **objeto**.



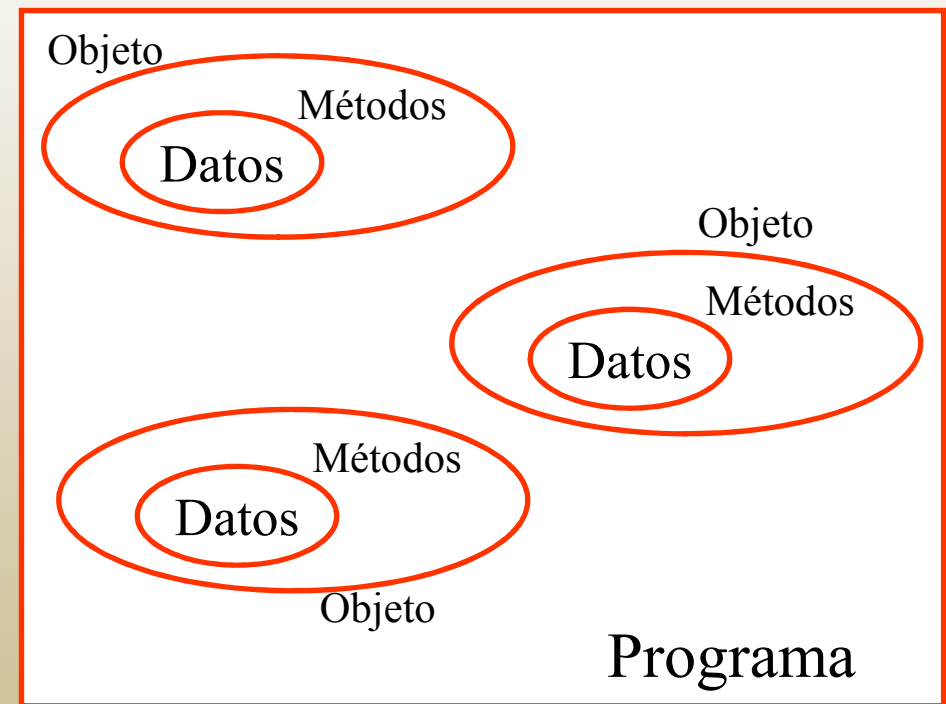
Un objeto es una unidad que contiene datos y operaciones que operan sobre esos datos. Los objetos de un sistema se comunican entre sí mediante mensajes.

# Un programa según....

## Programación Estructurada



## Programación Orientada a Objetos



Grady Booch resume la diferencia de la siguiente forma:

“Lea las especificaciones del sistema que desea construir. Subraye los verbos si persigue un código procedimental, o los sustantivos si su objetivo es un programa orientado a objetos”.

# Programación Orientada a Objetos

Todo lo que vemos a nuestro alrededor puede ser considerado un objeto (una computadora, un teléfono celular, un árbol, un automóvil, etc).

Ejemplo: una computadora está compuesta por varios componentes (tarjeta madre, chip, disco duro y otros), el trabajo en conjunto de todos ellos hace operar a una computadora. El usuario no necesita saber como trabajan internamente cada uno de estos componentes, sino como es la interacción con ellos. Es decir, cuando se conoce como interaccionan los componentes entre sí, el usuario podrá armar la computadora.



Relación con la Programación Orientada a Objetos

# Programación Orientada a Objetos

La Programación orientada a objetos trabaja de esta manera: todo el programa está construido en base a diferentes componentes (objetos), cada uno tiene un rol específico en el programa y todos los componentes pueden comunicarse entre ellos de formas predefinidas.



Todo objeto del mundo real tiene dos partes características y comportamiento.

## Automóvil

### **Características:**

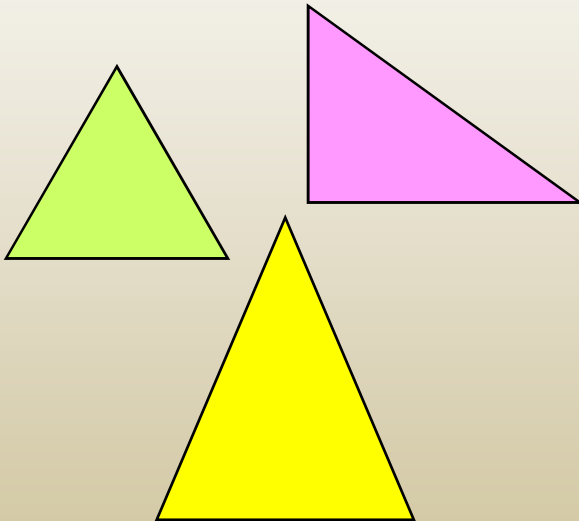
marca, modelo, color, velocidad máxima, velocidad actual, tamaño tanque de combustible, cantidad actual de combustible en el tanque.

### **Comportamiento:**

frenar, acelerar, llenar tanque de combustible, etc

# Programación Orientada a Objetos

Tienen algo en común?





# Programación Orientada a Objetos -CLASES

Clase Auto;

Variables  Son privadas no accesibles desde afuera.

Métodos  Son privados o públicos.

End;

## Sintaxis

Clase nombreClase;

    características

    comportamiento

End;

# Introducción – Programación Orientada a Objetos -CLASES

Clase Auto;

marca: string  
modelo: string  
color: string  
velocidad: integer;  
capacidad baúl: real



Siempre debe existir

constructor crear (unaMarca,unModelo,unColor:string; unaCapacidad:real)  
procedure arrancar;  
procedure acelerar(vel:integer);  
procedure desacelerar(vel:integer);  
procedure apagar;  
procedure verColor(); {*todos los otros!!!!*}  
procedure pintar(nuevoColor: string);

End;

Se deben implementar cada uno de los métodos definidos

# Introducción – Programación Orientada a Objetos - CLASES

Clase Auto;

marca: string; modelo: string  
color: string; velocidad: integer;  
capacidad baúl: real;

**constructor** crear (unaMarca,unModelo,unColor:string; unaCapacidad:real)

Begin

    marca:= unaMarca;   modelo:= unModelo;   color:= unColor;   capacidad:= unaCapacidad;

End;

**procedure** arrancar;

Begin

    velocidad:= 0;

End;

**procedure** acelerar(vel:integer)

Begin

    velocidad:= velocidad + vel;

End;

Notar que a diferencia  
de los TADs el objeto  
no es pasado como  
parámetro

# Introducción – Programación Orientada a Objetos - CLASES

```
procedure arrancar;  
Begin  
  velocidad:= 0;  
End;
```

```
procedure acelerar(vel:integer)  
Begin  
  velocidad:= velocidad + vel;  
End;
```

```
Procedure verColor(var unColor: string)  
Begin  
  unColor:=color;  
End;
```

```
procedure pintar (nuevoColor: string);  
Begin  
  color:= nuevoColor;  
End;
```

Notar que a diferencia  
de los TADs el objeto  
no es pasado como  
parámetro

# Introducción – Programación Orientada a Objetos -CLASES



Program uno;

Var

a1,a2: Auto;

Begin

a1:= Auto.crear (“mercedes”,”claseA”,”azul”, 150.36)

a1.arrancar;

a1.acelerar (100);

a1.desacelerar(30);

a2:= Auto.crear (“ferrari”,”2007”,”roja”, 0);

a2.arrancar;

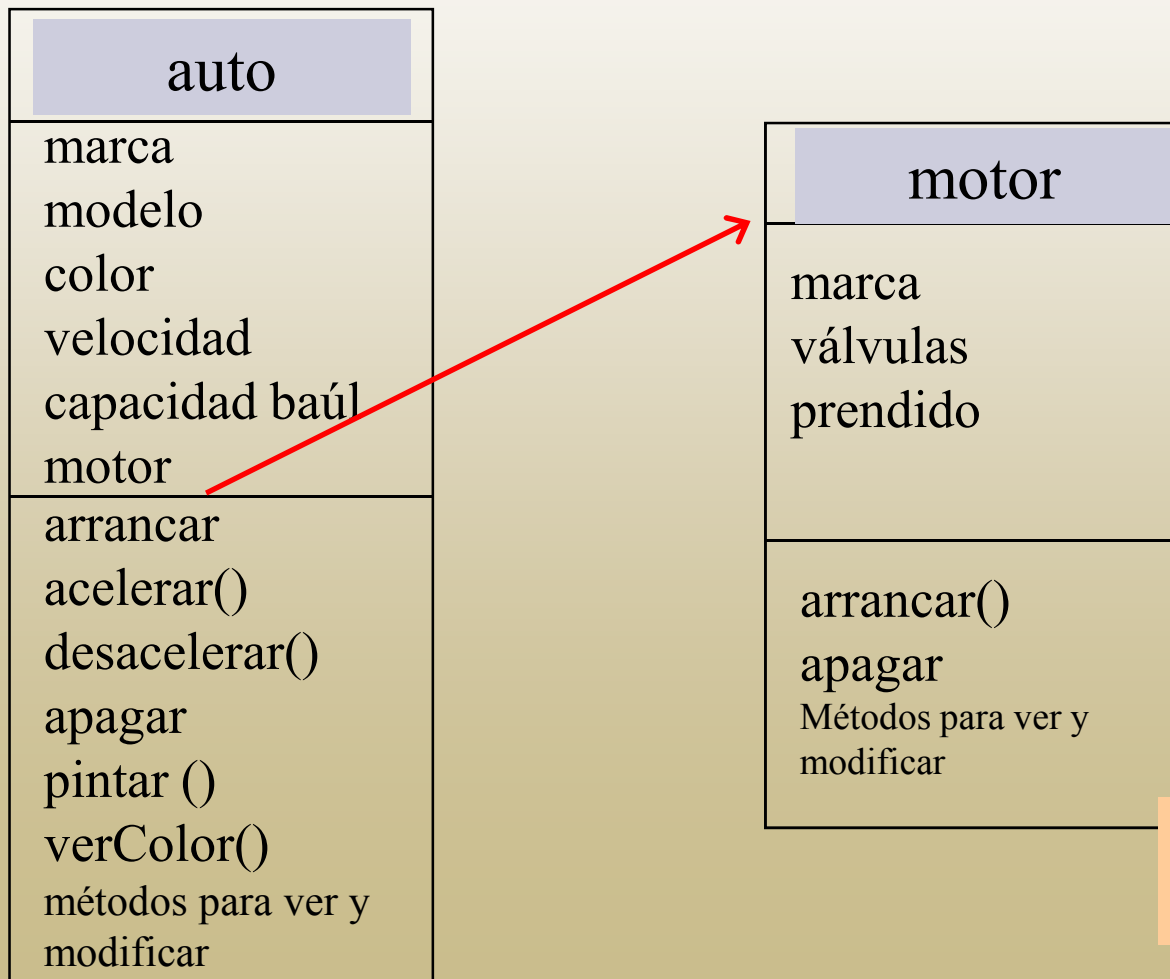
End.

Notar que sólo para utilizar la operación “constructor” debe anteponerse el nombre de la clase.

Notar que a diferencia de los TADs las operaciones se identifican como varObjeto.método (salvo el constructor)

## Introducción – Programación Orientada a Objetos

Considerando la clase auto descripta anteriormente. Además podemos suponer que el auto está compuesto por un motor, el cual podría estar representado por otra clase motor



Cómo redefinimos el  
auto?

Clase auto;

marca: string; modelo: string  
color: string; velocidad: integer;  
capacidad baúl: real; **miMotor: motor;**

constructor crear (unaMarca, unModelo, unColor: string; unaCapacidad: real; ;  
**marcaMotor: string; valvulasMotor: integer)**

Begin

marca:= unaMarca; modelo:= unModelo;  
color:= unColor; capacidad:= unaCapacidad;

**miMotor:= Motor.crear(marcaMotor, valvulasMotor);**

End;

procedure arrancar(vel: integer)

Begin

velocidad:= 0;

**miMotor.arrancar;**

End;



```
procedure apagar;  
Begin  
  velocidad:= 0;  
  miMotor.apagar;  
End;
```

```
procedure acelerar(vel:integer)  
Begin  
  velocidad:= velocidad + vel;  
End;
```

```
procedure pintar(unColor:string)  
Begin  
  color:= unColor;  
End;
```

```
procedure desacelerar(vel:integer)  
Begin  
  velocidad:= velocidad - vel;  
End;
```

```
procedure verColor(var unColor:string)  
Begin  
  unColor:= color;  
End;
```

Clase motor;

```
marca: string; valvulas:integer;  
prendido:boolean;
```

```
constructor crear (unaMarca:string; unaValvula:integer)
```

```
Begin
```

```
    marca:= unaMarca;  
    valvulas:= unaValvula;  
    prendido:= false;
```

```
End;
```

```
procedure arrancar
```

```
Begin
```

```
    prendido:= true;
```

```
End;
```

```
procedure apagar
```

```
Begin
```

```
    prendido:= false;
```

```
End;
```

Cómo hacemos el  
programa que lo usa?



```
Program dos;  
Var  
  a1,a2: Auto;  
Begin  
  a1:= Auto.crear("mercedes","claseA","azul", 150.36, "mercedes",16);  
  a1.arrancar;  
  a1.acelerar(30);  
  a1.apagar;  
End.
```

Cómo funciona?

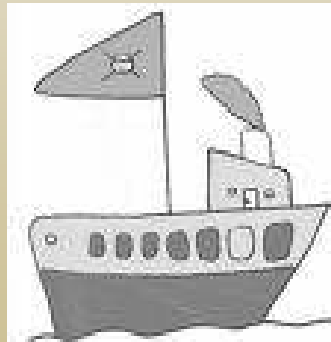
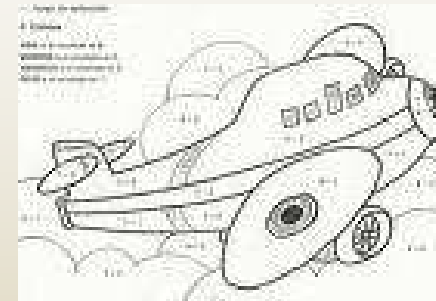
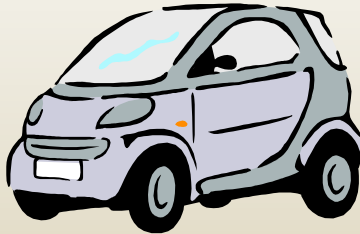
# Programación Orientada a Objetos

El modelo orientado a objetos consta de 4 conceptos básicos:

{  
Objetos  
Clases  
Herencia  
Envío de Mensajes

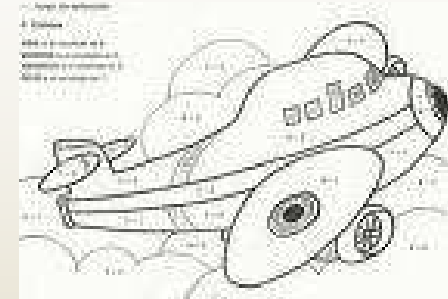
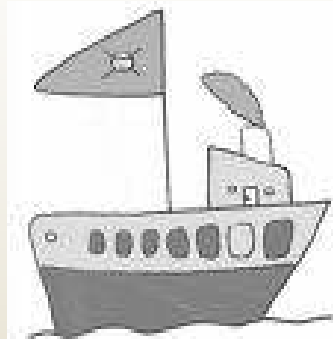
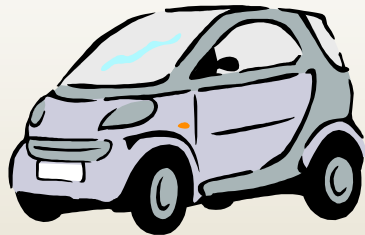
# HERENCIA – Conceptos

Qué características tienen en común?



Son medios de transporte  
Tienen marca  
Color  
Modelo  
Motor

# CLASES – Características



Marca  
Modelo  
Color  
Velocidad  
Motor

Marca  
Modelo  
Color  
Velocidad  
Motor

Marca  
Modelo  
Color  
Velocidad  
Motor

posiciónAncla

trenAterrizaje

Capacidad del baúl

Potencia de las turbinas  
Eslora

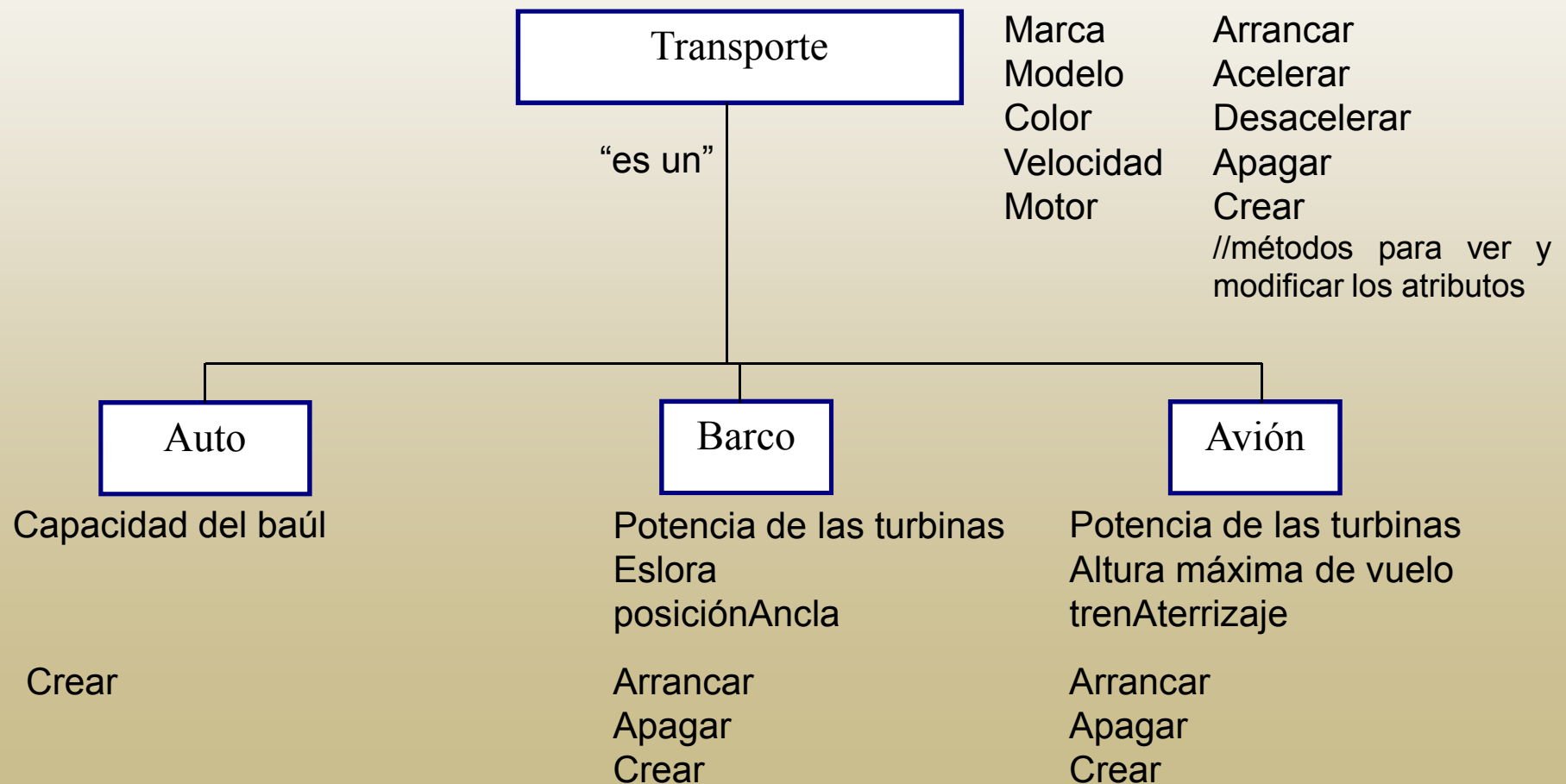
Potencia de las turbinas  
Altura máxima de vuelo  
Tamaño de las alas

Qué se puede notar?

Hay características comunes y propias de cada uno

# HERENCIA - Ejemplo

Cuando ocurren este tipo de cosas aparece el concepto de HERENCIA



- La *herencia* es el mecanismo que le permite a un objeto heredar propiedades de otra clase de objetos. La herencia permite a un objeto contener sus propios procedimientos o funciones y heredar los mismos de otros objetos.
- Un mecanismo potente que no se encuentra en sistemas procedimentales.
- La herencia hace las tareas de programación más fáciles, ya que se pueden crear sus objetos de modo creciente. Es decir, se puede definir un tipo general de clase y se utiliza como una parte de objetos específicos sin necesidad de tener que declarar todos los campos individuales nuevamente.
- Para definir una clase que hereda de otra clase se debe incluir el nombre de la clase “padre” entre paréntesis.

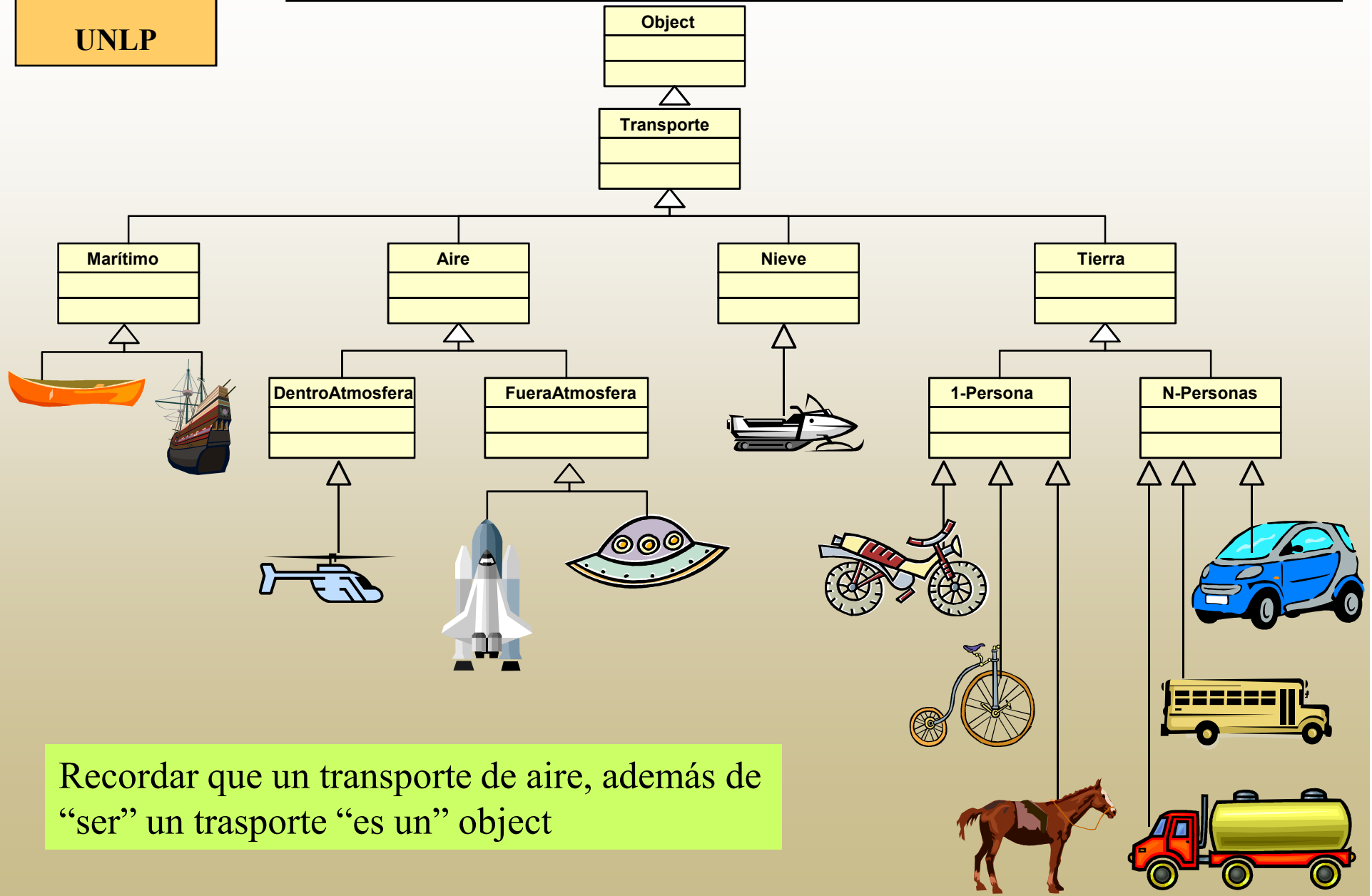
Así ocurre en nuestro ejemplo de los transportes. Dónde vemos la herencia allí?



- Por ejemplo, la subclase auto, barco y la subclase avión heredan todos los métodos y atributos correspondientes a los transportes, por ser estas subclases de la clase transporte.
- Además, al crear un objeto auto, tendrá no sólo los atributos y comportamiento propios de un transporte sino también los específicos de un auto, por ejemplo podré conocer la capacidad del baúl.

- La herencia consiste entonces en utilizar una clase ya creada para tomar sus características en clases más **especializadas o derivadas** de ésta para reutilizar el código que sea común con la clase base y solamente *definir* nuevos métodos o *redefinir* algunos de los existentes.
- Debido a la herencia, los programas orientados a objetos constan de taxonomías, árboles y jerarquías de clases que, por medio de la **subclasificación**, llegan a ser más *específicas*.
- Existe la clase OBJECT la cual es “super” clase de todas las clases que se definen en la aplicación. Es la clase padre por defecto si en la definición de la nueva clase no se especifica otra.

# Jerarquía de Clases



Recordar que un transporte de aire, además de “ser” un transporte “es un” object

## HERENCIA - Consideraciones

Es importante hacer notar que la programación orientada a objetos incluye el concepto de herencia, el cual no es incluido en la programación estructurada.



Notar que no es lo mismo la herencia en los objetos a que un Tad utilice a otro Tad (por ejemplo el Tad persona podría utilizar el Tad fecha), ya que eso NO significa que el Tad persona HEREDE las características y comportamiento del Tad fecha.

Cómo aplicamos la herencia en nuestro ejemplo de transportes?

- Cuántas clases definimos?
- Dónde definimos cada atributo?
- Dónde definimos cada método?
- Dónde implementamos cada método?

Cuántas clases definimos?

↳ La clase transporte, auto, barco, avión

Dónde definimos cada atributo?

↳ Los atributos comunes a todas las clases en la clase “padre” (Transporte), y los particulares de cada clase en cada una de ellas (Auto, Barco, Avión).

Dónde definimos cada método?

↳ Los métodos comunes en la clase “padre” y los correspondientes a los atributos propios de cada clase en cada una de ellas. Además un método puede definirse en la clase hijo y padre a la vez.

Dónde implementamos cada método?

↳ Depende de que queramos implementar, ya lo veremos

Clase Transporte;

```
marca: string; modelo: string  
color: string; velocidad: integer;  
miMotor: motor;  
constructor crear (unaMarca, unModelo, unColor: string;  
                   marcaMotor: string; valvulasMotor: integer)
```

Begin

```
marca:= unaMarca;  modelo:= unModelo;  
color:= unColor;  
miMotor:= Motor.crear(marcaMotor, valvulasMotor);
```

End;

```
procedure arrancar(vel: integer)
```

Begin

```
velocidad:= 0;  
miMotor.arrancar;
```

End;

## Ejemplo - HERENCIA

```
procedure desacelerar(vel:integer)
Begin
    velocidad:= velocidad - vel;
End;

procedure apagar;
Begin
    velocidad:= 0;
    miMotor.apagar;
End;

procedure acelerar(vel:integer)
Begin
    velocidad:= velocidad + vel;
End;

procedure pintar(unColor:string)
Begin
    color:= unColor;
End;
```

Cómo  
implementamos la  
clase auto?.

## Ejemplo - HERENCIA

Clase Auto (**Transporte**);

Indica que la clase auto hereda de la clase transporte (se indica entre paréntesis)

capacidad baúl: real;

Sólo se definen las características propias del auto

constructor crear (unaMarca,unModelo,unColor:string; unaCapacidad:real;  
marcaMotor:string; valvulasMotor:integer)

Begin

capacidad:= unaCapacidad;

super.crear(unaMarca,unModelo,unColor,marcaMotor,valvulasMotor);

End;

Invoca al método crear de la clase “padre” (de la cual hereda)

Sólo se implementan los métodos propios del auto

Notar que los métodos no implementados se heredan de la clase padre



## Ejemplo - HERENCIA

Clase Barco (**Transporte**);

```
potenciaTurbinas: real;  
eslora: real;  
posicionAncla: string;
```

```
constructor crear (unaMarca, unModelo, unColor: string; unaPotencia: real;  
                  unaEslora: real; marcaMotor: string; valvulasMotor: integer)
```

Begin

```
potenciaTurbinas:= unaPotencia;  
eslora:= unaEslora;  
posicionAncla:= “en superficie”;  
super.crear(unaMarca, unColor, marcaMotor, valvulasMotor);
```

End;

Procedure arrancar

Begin

```
posicionAncla:= “en superficie”;  
super.arrancar;
```

End;

Procedure apagar

Begin

```
posicionAncla:= “en agua”;  
super.apagar;
```

End;

Notar que los métodos no implementados se heredan de la clase padre

Notar que se reimplementaron los métodos arrancar y apagar ya que se les quiere dar un comportamiento diferente

## Ejemplo - HERENCIA

Clase Avion (**Transporte**);

```
potenciaTurbinas: real;  
alturaMaxVuelo:real;  
trenAterrizaje:string;
```

```
constructor crear (unaMarca,unModelo,unColor:string; unaPotencia:real;  
                  unaAltura:real;marcaMotor:string; valvulasMotor:integer)
```

Begin

```
potenciaTurbinas:= unaPotencia;  
alturaMaxVuelo:= unaAltura;  
trenAterrizaje:=“desplegado”;  
super.crear(unaMarca,unColor,marcaMotor,valvulasMotor);
```

End;

Procedure arrancar

Begin

```
trenAterrizaje:=“ no desplegado”;  
super.arrancar;
```

End;

Procedure apagar

Begin

```
trenAterrizaje:=“desplegado”;  
super.apagar;
```

End;

Notar que los métodos no implementados se heredan de la clase padre

Notar que se reimplementaron los métodos arrancar y apagar ya que se les quiere dar un comportamiento diferente

## Ejemplo - HERENCIA

Program dos;

Var

a:auto; b:barco, av:avion;  
ma,mo,co,maMotor:string;  
val:integer;

Begin

read (mo,ma,co,maMotor);  
read(val)

a:= Auto.crear (ma, mo,co,150.23,maMotor,val);

a.arrancar;

a.acelerar (30);

b:= Barco.crear ("Royal", "Nautilus", "verde", 200.90, 7.80, "Honda", 5);

b.arrancar;

b.acelerar(50);

b.pintar("rojo");

a.apagar;

End.

Se llaman igual



Recordar que cuando un objeto invoca a un método, primero se lo busca en su definición, sino está definido en él, se lo busca en el "padre" del objeto.

## HERENCIA - Consideraciones

Cuando un programador define una jerarquía de clases, es porque identifica características comunes en los objetos, y algunas características que los diferencian.

Los atributos comunes se definen en la “super” clase (en nuestro ejemplo la clase transporte).

Los atributos diferentes se definen en cada clase (en nuestro ejemplo la clase auto, barco, avión).

Los métodos que se implementan de igual manera para todas las clases, deben implementarse en la “super” clase.

Los métodos que se implementan de manera diferente en cada subclase, deben implementarse en cada una.

Para hacer referencia a un método de una super clase desde una subclase debe ponerse: `super.nombremetodo`.

Otros conceptos clave además de los vistos que resumen las ventajas de la programación orientada a objetos son:

- Encapsulamiento
- Abstracción
- Polimorfismo

- Es el término formal que describe al conjunto de métodos y datos dentro de un objeto de forma que el acceso a los datos se permite solamente a través de los propios métodos del objeto.
- Ninguna otra parte de un programa orientado a objetos puede operar directamente sobre los datos de un objeto.
- La comunicación entre un conjunto de objetos sucede exclusivamente por medio de mensajes explícitos. (Concepto que también poseen los TADs)

- La orientación a objetos fomenta que los programadores y usuarios piensen sobre las aplicaciones en términos abstractos.
- Comenzando con un conjunto de objetos, se busca un factor de comportamiento común y se sitúa en clases superiores.
- Las bibliotecas de clases proporcionan un depósito para elementos comunes y reutilizables.
- La herencia mantiene automáticamente las relaciones entre las clases dispuestas jerárquicamente en una biblioteca de clases.

- Cada nivel de abstracción facilita el trabajo de programación porque hay disponible más cantidad de código reutilizable.
- Podríamos decir que la abstracción es la capacidad de un objeto de cumplir sus funciones independientemente del contexto en el que se lo utilice.
- O sea, un cierto objeto siempre expondrá sus mismas propiedades y dará los mismos resultados a través de sus eventos, sin importar el ámbito en el cual se lo haya creado.



- Los objetos actúan en respuesta a los mensajes que reciben.
- El mismo mensaje puede originar acciones completamente diferentes al ser recibido por diferentes objetos. Este fenómeno se conoce como polimorfismo.

- El mensaje crear, por ejemplo, al ser enviado a un auto o barco invocará diferentes métodos de creación.
- El polimorfismo, entonces, se refiere a que una misma operación puede tener diferente comportamiento en diferentes objetos.

- En una aplicación orientado a objetos es posible definir colecciones cuyos elementos son genéricos.
- Al definir que los elementos de nuestra colección sean de tipo object, se permite almacenar cualquier objeto.
  - Recordar que toda clase hereda de la clase object.

## OBJETOS – Colecciones - Ejemplo

```
Clase Lista;  
  lis = ^nodo;  
  nodo = record  
    elemento:object;  
    sig:lis;  
  end;  
  pri: lis;  
  actual:lis;
```

Indica que el elemento de la lista es cualquier objeto.

```
Constructor crear;  
Procedure agregar (unObjeto:object)
```

Indica que se va a agregar un objeto cualquiera.

```
...  
Procedure iniciarRecorrido;  
....  
Procedure devolverElemento (var unObjeto:object)  
...  
Procedure avanzarRecorrido;  
...  
Function fin: boolean;  
...  
End;
```

Cómo se utiliza?

## OBJETOS – Colecciones - Ejemplo

Program uno;

Var

l: Lista[Auto];

a: Auto;

l2: Lista[Barco];

b: Barco;

Begin

a:= Auto.crear(.....);

b:= Barco.crear(.....);


l:= Lista.crear;

l.agregar(a);


l2:= Lista.crear;

l2.agregar(b);

End.



Indica que los elementos de la lista son instancias de la clase Auto.



Indica que los elementos de la lista son instancias de la clase Barco.