


# Programación Orientada a Objetos

Isidro González Caballero  
(Universidad de Oviedo)  
Técnicas de Comp. en Física  
Santander, 08/11/2010



# Introducción

- Los problemas suelen tener **varias soluciones** posibles.
- En programación existen **diversas metodologías** que nos ayudan a enfrentar un problema.
- Cada metodología tiene **diversos lenguajes** que las soportan.
  - Algunos lenguajes soportan varias metodologías.

Metodología	Lenguaje
Estructurada	Fortran, C, Pascal, Basic
<b>Orientada a objetos (OOP)</b>	<b>C++</b> , Java, Smalltalk
Orientada a eventos	VisualBasic

# Programación Orientada a Objetos

## Definición:

La **Programación Orientada a Objetos (OOP)** es un **método** de programación en el cual los programas se organizan en colecciones cooperativas de **objetos**, cada uno de los cuales representa una **instancia** de alguna **clase**, y cuyas clases son, todas ellas, miembros de una **jerarquía de clases** unidas mediante **relaciones de herencia**.

## Comentarios:

- Usamos **objetos** en lugar de **algoritmos** como bloque fundamental
- Cada objeto es una **instancia** de una clase
- Las clases están relacionadas entre sí por relaciones tan complejas como la **herencia**

# Ventajas de la POO

- Proximidad de los conceptos modelados respecto a objetos del mundo real
- Facilita la reutilización de código
  - Y por tanto el mantenimiento del mismo
- Se pueden usar conceptos comunes durante las fases de análisis, diseño e implementación
- Disipa las barreras entre el *qué* y el *cómo*

# Desventajas de la POO

- Mayor **complejidad** a la hora de entender el flujo de datos
  - Pérdida de linealidad
- Requiere de un lenguaje de modelización de problemas más elaborado:
  - *Unified Modelling Language* (UML)
  - **Representaciones gráficas** más complicadas

# Conceptos de la OOP

## Conceptos básicos

- Objeto
- Clase

## Características de la OOP

- Abstracción:
- Encapsulamiento:
- Modularidad:
- Jerarquía

## Otros conceptos OOP

- Tipos
- Persistencia

## Tipos de relaciones

- Asociación
- Herencia
- Agregación
- Instanciación

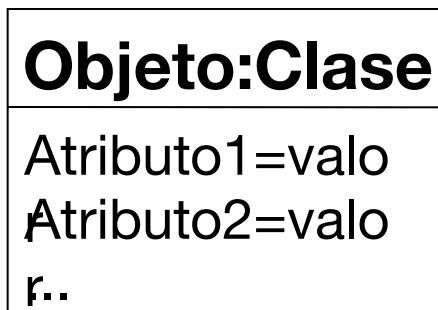
## Representaciones gráficas

- Diagramas estáticos (de clases, de objetos...)
- Diagramas dinámicos (de interacción...)

# Objeto y Clase

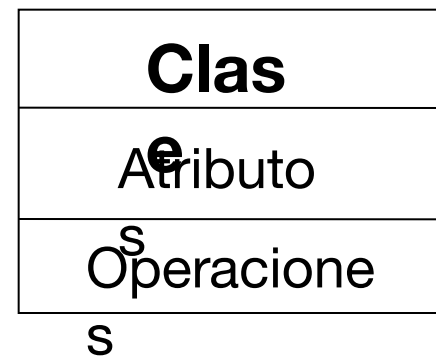
Un **objeto** es algo de lo que hablamos y que podemos manipular

- Existen en el mundo real (o en nuestro entendimiento del mismo)



Una **clase** describe los objetos del mismo tipo

- Todos los objetos son **instancias** de una clase
- Describe las **propiedades** y el **comportamiento** de un tipo de objetos

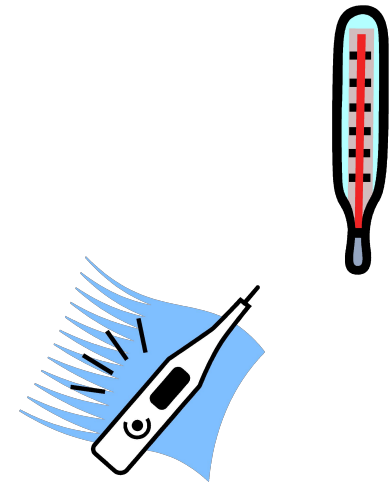


# Conceptos OOP: Abstracción

- Nos permite trabajar con la **complejidad del mundo real**
  - Resaltando los aspectos relevantes de los objetos de una clase
  - Ocultando los detalles particulares de cada objeto
- Separaremos el **comportamiento** de la **implementación**
- Es más importante **saber qué se hace** en lugar de **cómo se hace**:

## Un sensor de temperatura

- Se define porque...
  - mide la temperatura
  - nos muestra su valor
  - se puede calibrar...
- No sabemos... (no nos importa)
  - cómo mide la temperatura
  - de qué está hecho
  - cómo se calibra



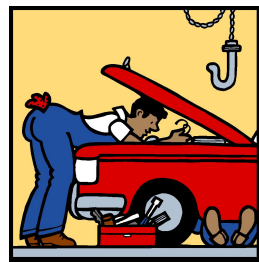


# Conceptos OOP: Abstracción

- La abstracción **no es única**:

## Un coche puede ser...

- Una cosa con ruedas, motor, volante y pedales (conductor)
- Algo capaz de transportar personas (taxista)
- Una caja que se mueve (simulador de tráfico)
- Conjunto de piezas (fabricante)



# Conceptos OOP: Encapsulamiento

- Ninguna parte de un sistema complejo debe depender de los detalles internos de otra.
- Complementa a la abstracción
- Se consigue:
  - Separando la interfaz de su implementación
  - Ocultando la información interna de un objeto
  - Escondiendo la estructura e implementación de los métodos (algoritmos).
  - Exponiendo solo la forma de interactuar con el objeto

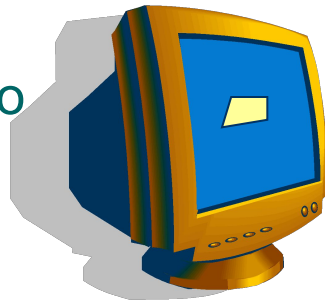
# Conceptos OOP: Encapsulamiento

## Ejemplo: Un paralelogramo



### Vemos que se puede...

- Construir con:
  - 4 puntos (y restricciones)
  - 1 punto y 2 vectores
  - 1 punto, 1 vector, 1 ángulo y 1 lado
- Transformaciones:
  - Escalado
  - Rotación
  - Desplazamiento
- Dibujar



### No vemos...

- Como está representado internamente
  - 4 puntos?
  - 1 punto y 2 vectores?
  - ...
- Como se modifica su escala
  - Guardando el factor?
  - Escalando en el momento?
- Idem para rotación, traslación, etc...

# Conceptos OOP: Modularidad

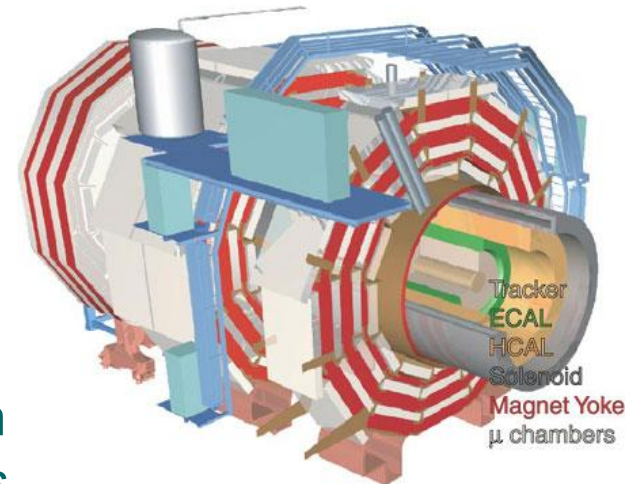
- Consiste en separar el sistema en bloques poco ligados entre sí: módulos.
  - Organización del código
- Es una especie de encapsulamiento de más alto nivel.
  - El C++ no lo impone aunque lo soporta (namespace)
  - El Java es más formal (packages)
- Difícil pero muy importante en sistemas grandes.
  - Suele aplicarse refinando el sistema en sucesivas iteraciones
  - Cada módulo debe definir una interfaz clara

# Conceptos OOP: Modularidad

## Ejemplo: Simulación detector de AAEE

Puede dividirse en los siguientes módulos...

1. **Geometría:** Describe el detector físicamente (forma, materiales, tamaño)
2. **Partículas:** Las partículas cuyas interacciones nos interesan
3. **Procesos:** Aquí enlazamos la información del detector (materia) con las propiedades de las partículas.
4. ...
  - Podríamos dividir el módulo de procesos en *procesos electromagnéticos*, *procesos hadrónicos*, ...
  - Lo mismo podríamos hacerlo con las partículas: *leptones*, *hadrones*, ...



# Conceptos POO: Jerarquía

- Es una **clasificación** u ordenamiento de las abstracciones
- Hay dos jerarquías fundamentales:
  - **Estructura de clases:**
    - Jerarquía “*es un/a*”
    - Relaciones de **herencia**
  - **Estructura de objetos:**
    - Jerarquía “*parte de*”
    - Relaciones de **agregación**
    - Está implementada de manera genérica en la estructura de clases

# Conceptos OOP: Jerarquía

## Ejemplo: Figuras planas y diagramas

- Una **figura plana** es:
  - Algo con una posición en el plano
  - Escalable
  - Rotable

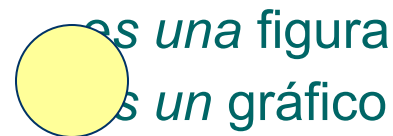
- Un **gráfico** es algo que se puede dibujar en 2D

- Un **diagrama** es un conjunto de cuadrados y círculos

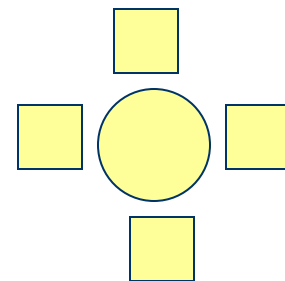
### Herencia simple

- Un cuadrado *es una* figura
- Un círculo *es una* figura

### Herencia múltiple



### Agregación



# Conceptos OOP: Tipo

- Es el **reforzamiento** del concepto de clase
- Objetos de tipo diferente no pueden ser intercambiados
- El C++ y el Java son lenguajes fuertemente “tipeados”
- Ayuda a corregir errores en tiempo de compilación
  - Mejor que en tiempo de ejecución



# Conceptos OOP: Persistencia

- Propiedad de un objeto de **trascender** en el tiempo y en el espacio a su creador (programa que lo generó)
- No se trata de **almacenar** sólo el estado de un objeto sino **toda la clase** (incluido su comportamiento)
- No está directamente soportado por el C++
  - Existen librerías y sistemas completos (OODBMS) que facilitan la tarea
  - Frameworks (entornos) como ROOT lo soportan parcialmente (reflex)
- El concepto de **serialización** del Java está directamente relacionado con la persistencia

# Relaciones

- Están presentes en cualquier sistema
- Definen como se producen los intercambios de información y datos
- También ayudan a comprender las propiedades de unas clases a partir de las propiedades de otras
- Existen 4 tipos de relaciones:
  - Asociación
  - Herencia
  - Agregación
  - Instanciación

# Relación de Asociación

- Relación más **general**
- Denota una **dependencia semántica**
- Es **bidireccional**
- **Primer paso** para determinar una relación más compleja

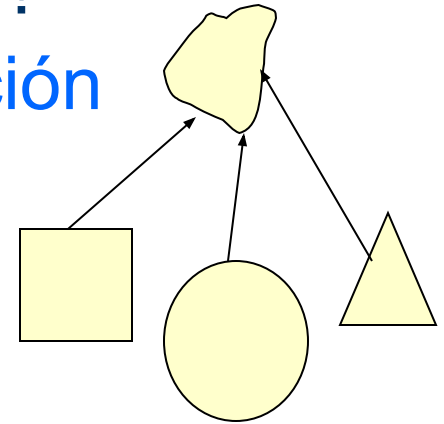
Ejemplo: Relación entre un producto y una venta. Cualquier venta está asociada a un producto, pero no es, ni forma parte de, ni posee ningún producto... al menos en una primera aproximación.

- **Cardinalidad:** multiplicidad a cada lado
  - Uno a uno: Venta-Transacción
  - Uno a muchos: Producto-Venta
  - Muchos a muchos: Comprador-Vendedor



# Relación de Herencia

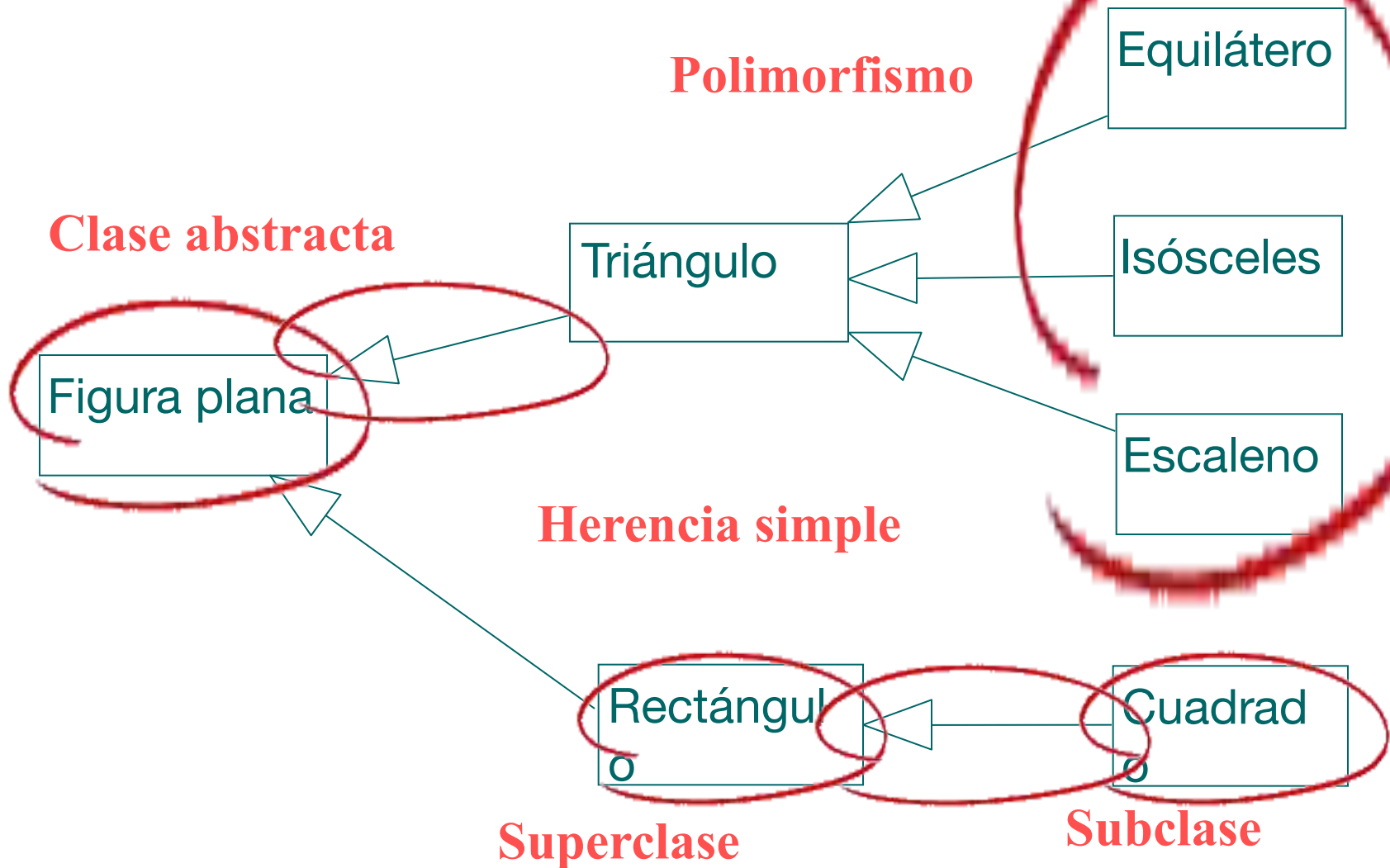
- ¡Relación **característica** de la OOP!
- Puede expresar tanto **especialización** como **generalización**
- Evita definir repetidas veces las **características comunes** a varias clases
- Una de las clases **comparte** la **estructura** y/o el **comportamiento** de otra(s) clase(s).
- También se denomina relación “*es un/a*” (*is a*)



# Relación de Herencia (vocabulario)

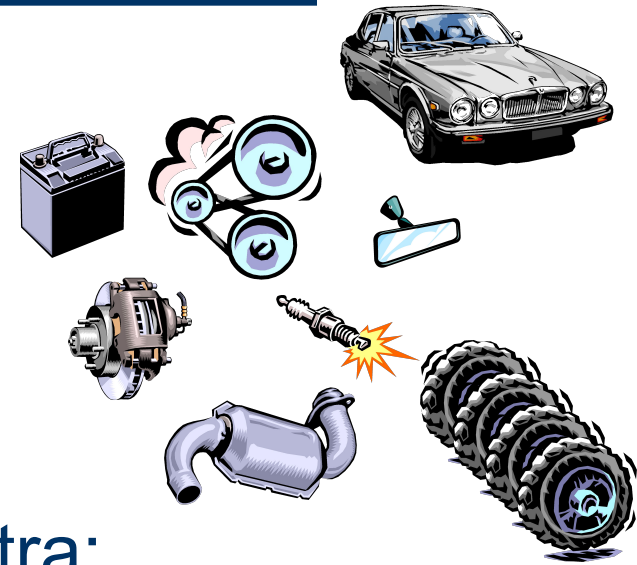
- **Clase base o superclase:** clase de la cual se hereda
- **Clase derivada o subclase:** clase que hereda
- **Herencia simple:** Hereda de una sola clase
- **Herencia múltiple:** Hereda de varias clases
  - Java solo la soporta parcialmente
  - Presenta diversos problemas (¿qué hacer cuando se hereda más de una vez de la misma clase?)
- **Clase abstracta:** La que no lleva, ni puede llevar, ningún objeto asociado
- **Polimorfismo:** Posibilidad de usar indistintamente todos los objetos de un clase y derivadas.

# Relación de Herencia (ejemplo)



# Relación de Agregación

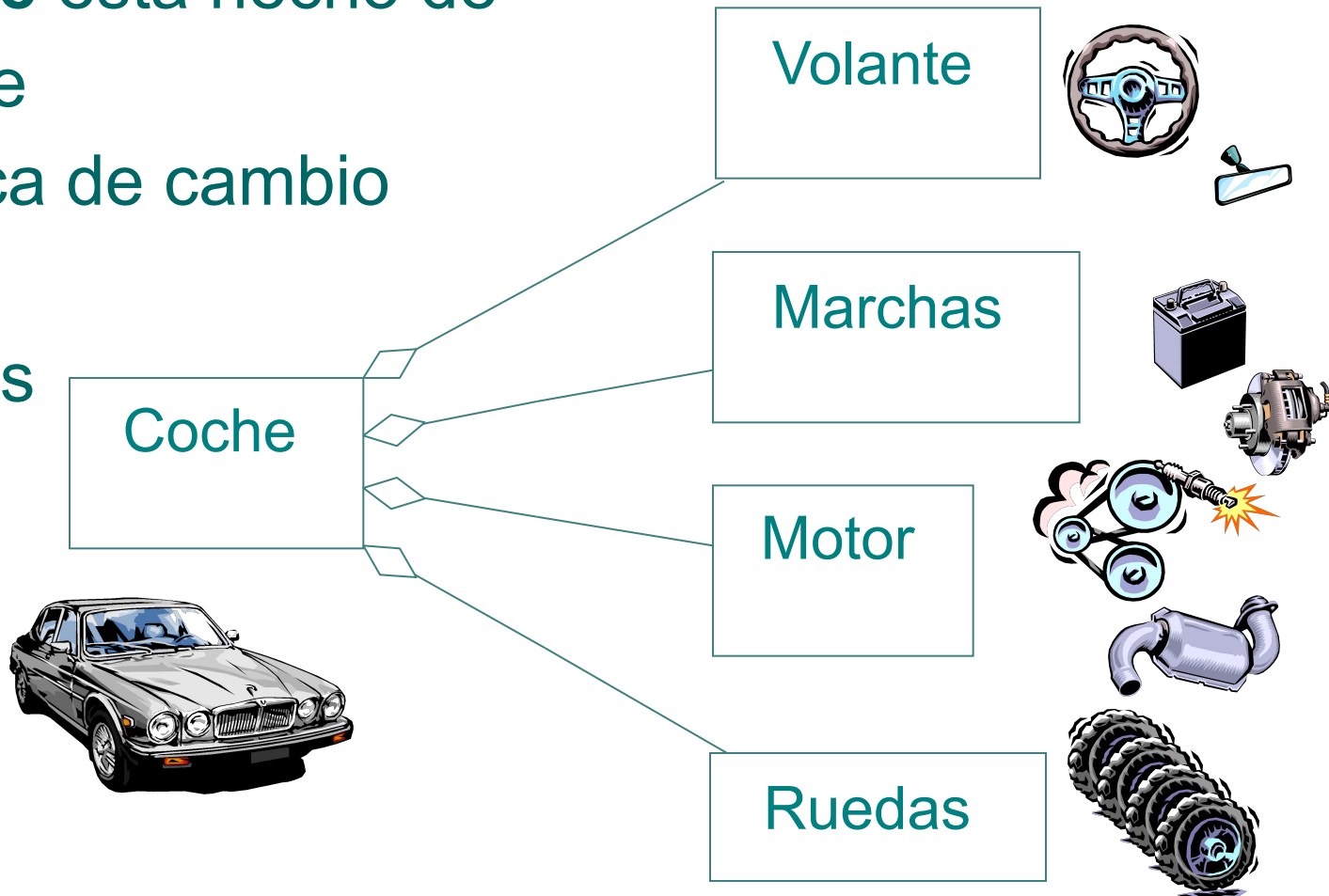
- Una **clase contiene** a otra **clase**
  - Ésta “es parte de” aquélla.
- También se denomina relación “*es parte de*” (has a)
- Una clase puede contener a otra:
  - **Por valor**: Cuando los objetos de la clase contenida se crean y destruyen al mismo tiempo que los de la clase continente
  - **Por referencia**: Cuando no necesariamente ocurre lo anterior



# Relación de Agregación

Un **coche** está hecho de

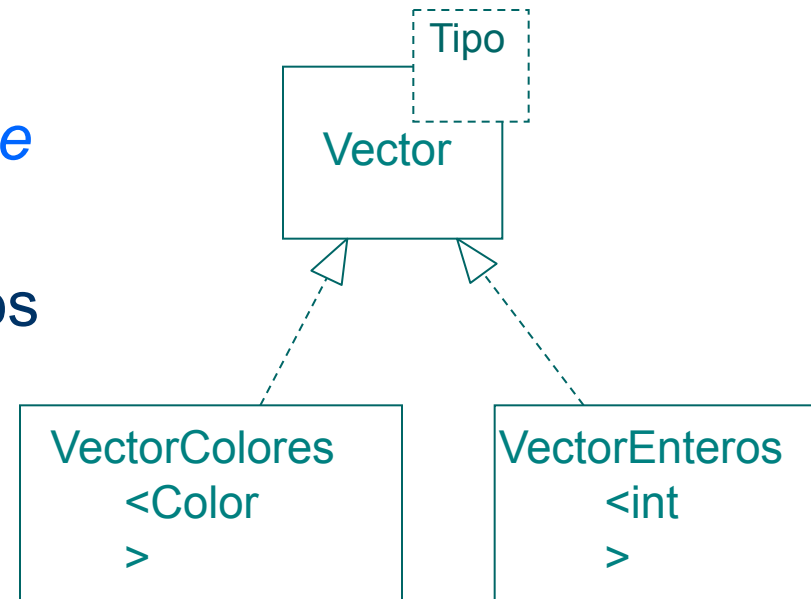
- Volante
- Palanca de cambio
- Motor
- Ruedas





# Relación de Instanciación

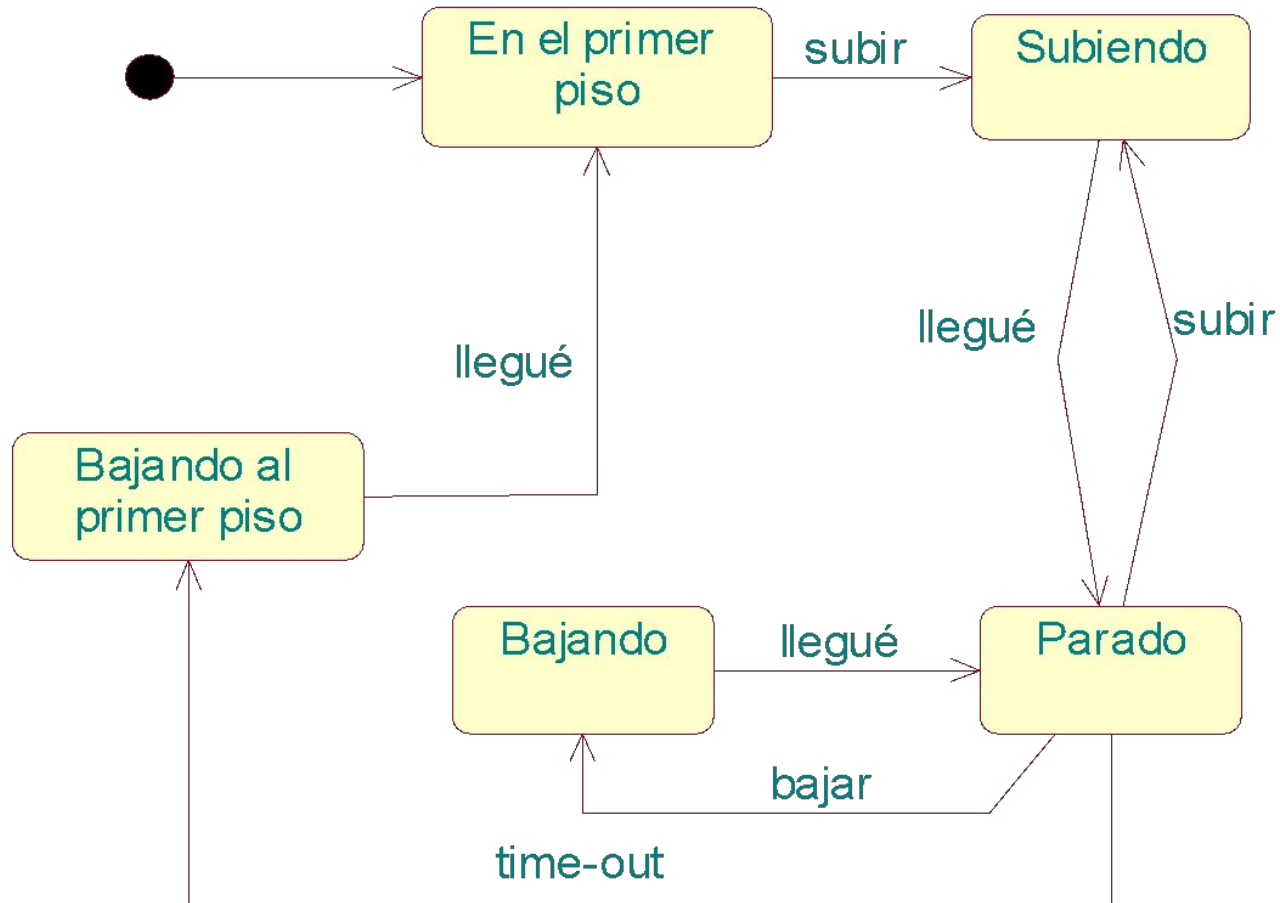
- En determinados casos una clase (p.ej. un vector) puede implementarse independientemente del tipo (real, complejo, color...) de alguno de sus atributos:
  - Definimos una **clase parametrizada** o *template* (plantilla)
  - Para cada uno de los tipos que necesitamos definimos una nueva clase ⇒ **Instanciación**



# Representaciones gráficas

- Nos sirven para comunicarnos con otros usuarios o desarrolladores.
- Documentan nuestro sistema
- Hay múltiples vistas y tipos de diagramas:
  - Estáticos
    - Diagramas de clases ⇨ Los de los ejemplos
    - Diagramas de objetos
    - ...
  - Dinámicos:
    - Diagramas de estado: Muestra el ciclo de vida de los objetos, sistemas, etc...
    - Diagramas secuenciales: Muestra como los objetos interaccionan entre ellos
    - ...

# Diagrama de estado: Un ascensor



# Diagrama secuencial: Impresión

