# Unidad 6 (III)

Diagramas de Clases

# Introducción a la visión orientada a objetos

- Vivimos en un mundo de objetos:
  - En la naturaleza: piedra, hoja...
  - En construcciones humanas: mesa, coche...
  - En los negocios: contrato, factura...

  etc
- Estos objetos pueden ser creados, modificados, destruidos, descritos, ordenados...
- Por este motivo se propone una visión orientada a objetos para la creación de software, una abstracción que modela el mundo real de forma que nos ayuda a entenderlo y gobernarlo mejor.

# Desarrollo de software orientado a objetos

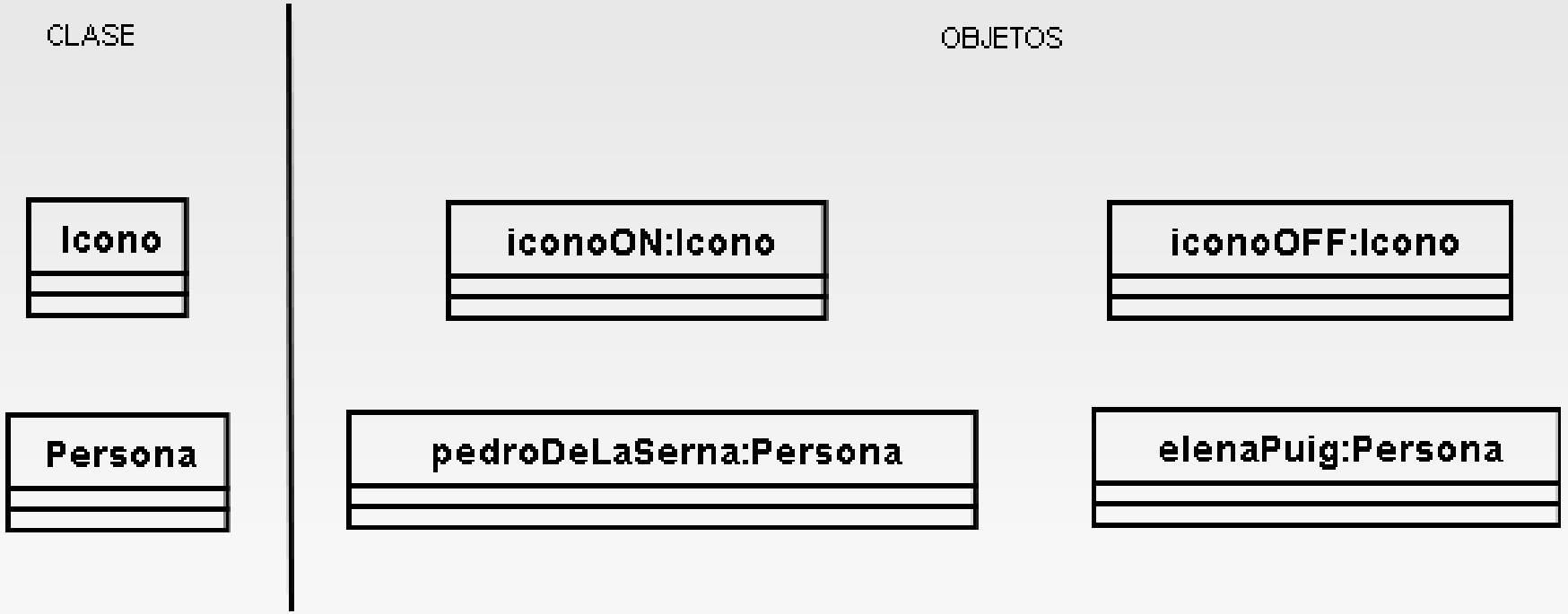
- El dominio del problema se caracteriza mediante un conjunto de objetos con atributos y comportamientos específicos.
- Los objetos se clasifican mediante clases.
- Suelen ser necesarias varias iteraciones para identificar todas las clases de un problema (proyecto, sistema).
- Las tecnologías de objetos facilitan la reutilización.
- Los sistemas orientados a objetos son más fáciles de mantener, de adaptar y de escalar.

# Clases y objetos

- Un objeto es un concepto, abstracción o cosa con significado a efectos del problema que se esté tratando. Es simplemente algo que tiene sentido en el contexto de la aplicación.
  - Ejemplo: “Icono de ON”, “Icono de OFF”, “Ventana de Error”, “Ventana de Aviso”, etc...
- Una clase describe una colección de objetos similares.
  - Ejemplo: “ICONO”, “VENTANA”...

# Clases y objetos (II)

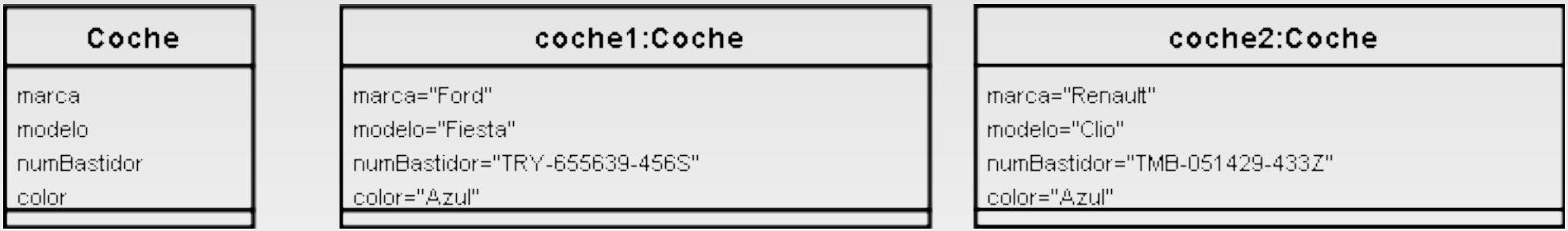
- Se dice que un objeto es una **instancia** de una clase.



# Atributos

- Un atributo es un valor de un dato que está almacenado en los objetos de una clase. Los atributos “describen” el objeto.
  - Ejemplo: La clase *Persona* puede tener los atributos **nombre**, **edad** y **profesión**.
- Cada atributo tiene un valor para cada instancia de la clase.
- El nombre del atributo es único dentro de la clase.

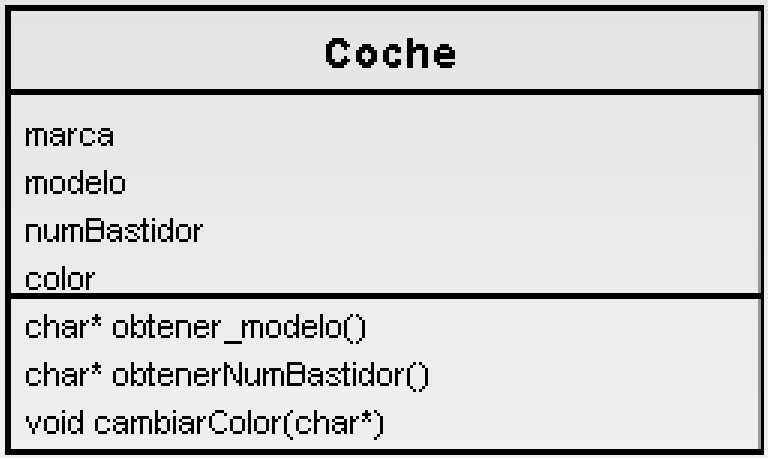
# Ejemplos de atributos



# Operaciones o métodos.

- Una operación (también llamada método) es una función o transformación que puede aplicarse a los objetos de una clase.
  - Ejemplo: La clase *Empresa* podría tener como métodos: ***contratar***, ***despedir***, ***repartir dividendos***.
- Por lo general, las operaciones actuarán sobre algunos de los atributos del objeto.

# Ejemplos de operaciones



# Responsabilidades

- Son las obligaciones que tienen las clases, de lo que se tienen que encargar.
- Ejemplos:
  - Una clase “Pared” puede tener como responsabilidades saber sobre su altura, anchura y grosor.
  - Una clase “Sensor” debe responsabilizarse de medir la temperatura y avisar cuando alcance un punto determinado.
- Al refinar el modelo, las responsabilidades se traducen en el conjunto de atributos y operaciones que las satisfacen

# Características de los objetos

- **Identidad:** Cada objeto posee su propia identidad inherente. Es decir, dos objetos serán distintos aunque los valores de todos sus atributos sean idénticos
- **Clasificación:** Los objetos con la misma estructura de datos (atributos) y comportamiento (operaciones) se “agrupan” para formar una clase.

# Características de los objetos (II)

- **Herencia:** Las clases se pueden organizar jerárquicamente, de forma que una clase (subclase) heredará todos los atributos y operaciones asociadas a su clase padre (superclase)
  - Ejemplo: La clase *Polígono* puede tener tres subclases: *Círculo, Triángulo, Rectángulo.*
- **Polimorfismo:** Una implementación de una operación en una subclase redefine la implementación de la misma operación en la superclase.
  - Ejemplo: Método **obtener_área()** de las clases Círculo, Triángulo y Rectángulo.

# Características de los objetos (III)

- **Encapsulación:** Los objetos encapsulan los datos y las operaciones que manipulan dichos datos. La única forma de operar sobre los atributos es a través de los métodos.
  - Los detalles de implementación interna de datos y métodos están ocultos al mundo exterior (**ocultación**). Esto reduce la propagación de efectos colaterales cuando se realizan cambios.
  - Al mezclar las estructuras de datos y las operaciones que las manipulan en una entidad sencilla (la clase) se facilita la reutilización de componentes.

# Características de los objetos (IV)

- **Abstracción:** Las clases y objetos identifican las características esenciales de algo, y omiten los detalles que no son importantes desde cierto punto de vista.
  - p.ej: En un sistema de nóminas, la clase “Empleado” tendrá un atributo “cuenta corriente”. Pero en un sistema que se encarga de asignar diversas tareas a los empleados, será innecesario que la clase “Empleado” contenga un atributo “cuenta corriente”.

# Modelado del vocabulario de un sistema

- Identificar aquellas cosas que utilizan los usuarios o programadores para describir el problema o la solución.
- Para cada abstracción, identificar un conjunto de responsabilidades.
- Hay que asegurarse de que cada clase esté claramente definida, y que hay un buen reparto de responsabilidades entre todas ellas.
- Hay que proporcionar a cada clase los atributos y operaciones necesarios para cumplir las responsabilidades.

# Diagramas de clases e instancias

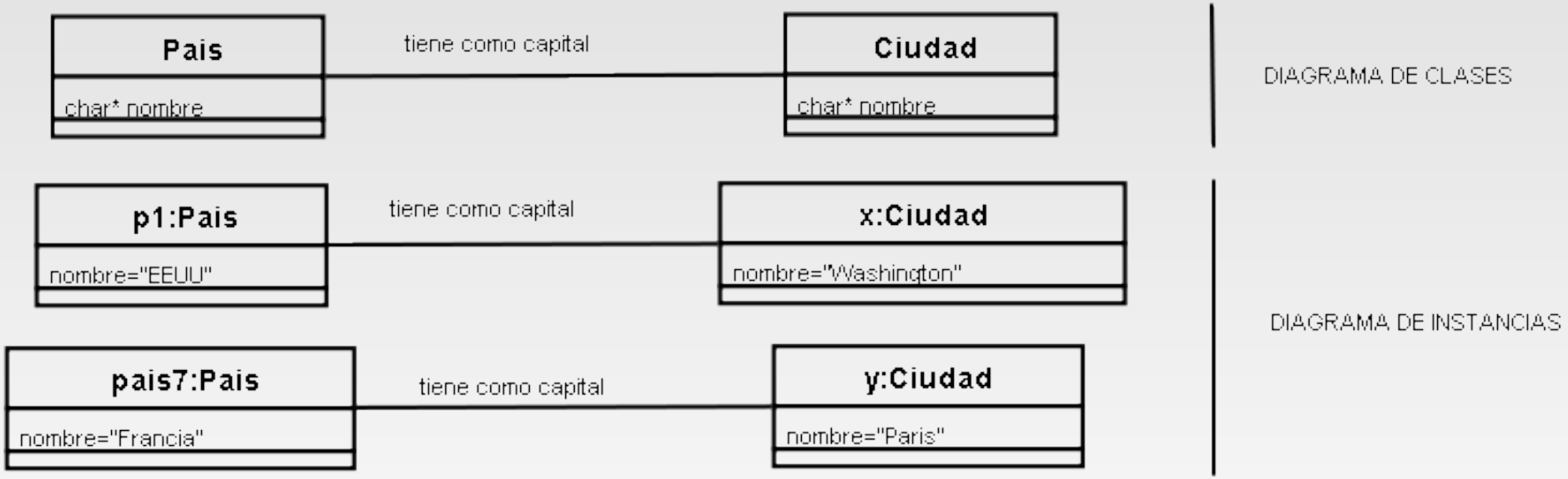
- **Diagramas de clases:** Describen clases de objetos.

- **Diagramas de instancias (u objetos):** Describen instancias de clases de objetos.

# Enlaces y asociaciones

- Un enlace es una conexión física o conceptual entre instancias de objetos.
- Una asociación describe un grupo de enlaces con estructura y semántica comunes. Es decir, los enlaces son instancias de las asociaciones.
- Las asociaciones son bidireccionales.

# Ejemplo asociaciones y enlaces



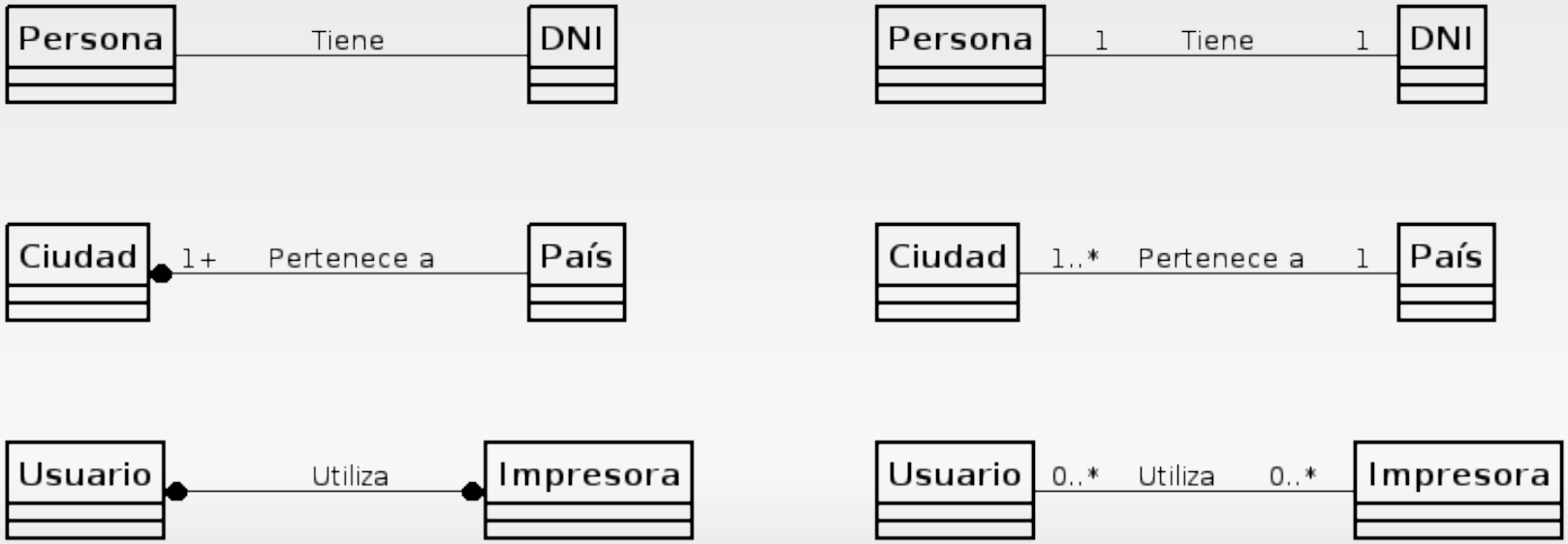
# Multiplicidad

- Número de instancias de una clase que pueden estar relacionadas con una única instancia de una clase asociada.

| | |
|---|---|
| 0..1 | Cero o una instancia |
| 0..* ó * | Cero o más instancias |
| 1 | Una instancia |
| 1..* | Una o más instancias |
| 4, 8 | Cuatro u ocho instancias |
| 6..12 | Entre seis y doce instancias |

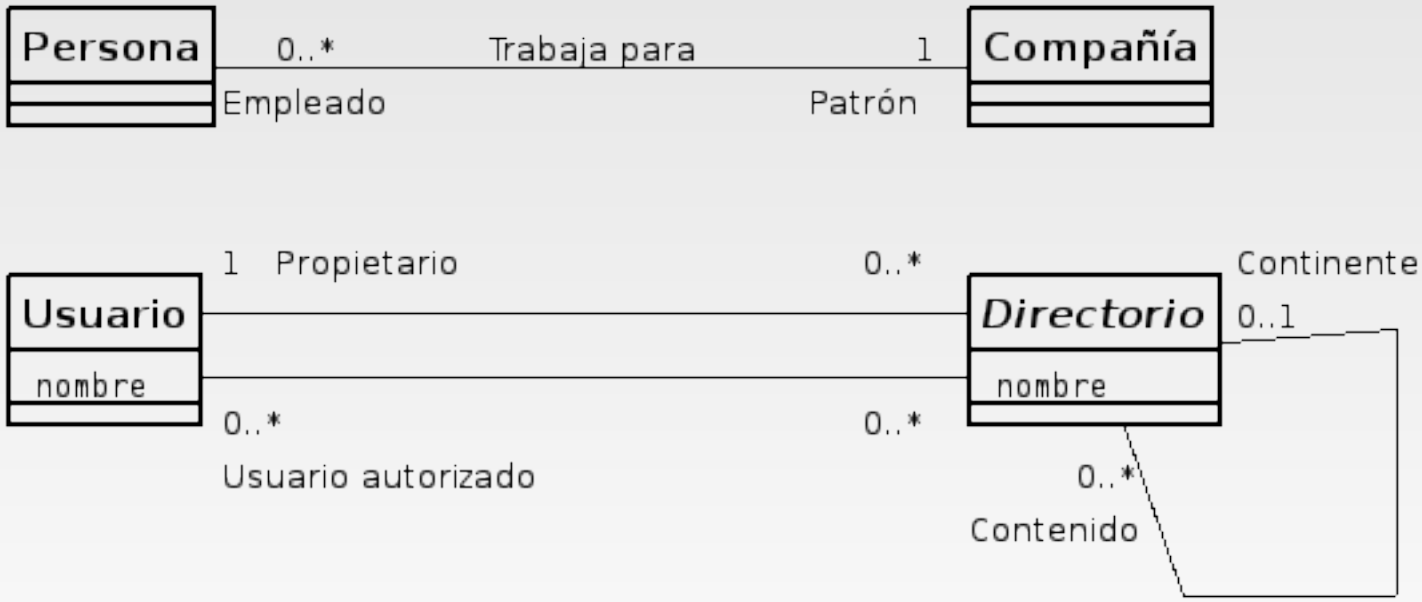
# Ejemplos de multiplicidad

- A veces se usa un círculo negro en el extremo de la asociación para expresar la multiplicidad 0..* y un círculo blanco para expresar 0..1



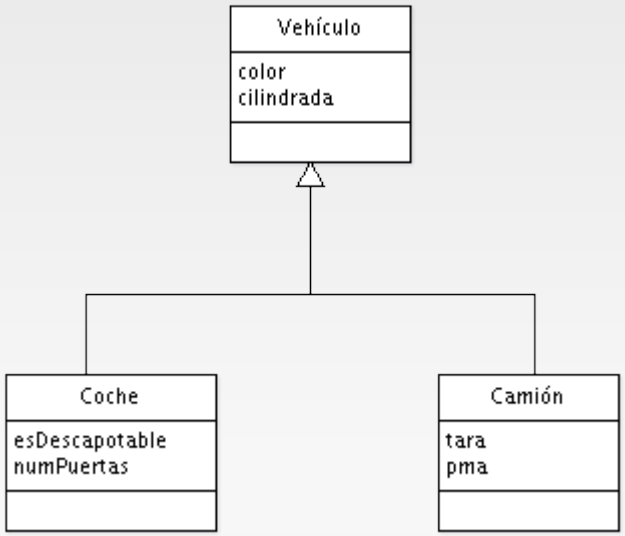
# Nombres de rol

- Es un nombre que identifica de forma única un extremo de una asociación



# Generalización y herencia

- Relación jerárquica entre clases.
- La subclase hereda los atributos y métodos de la superclase.

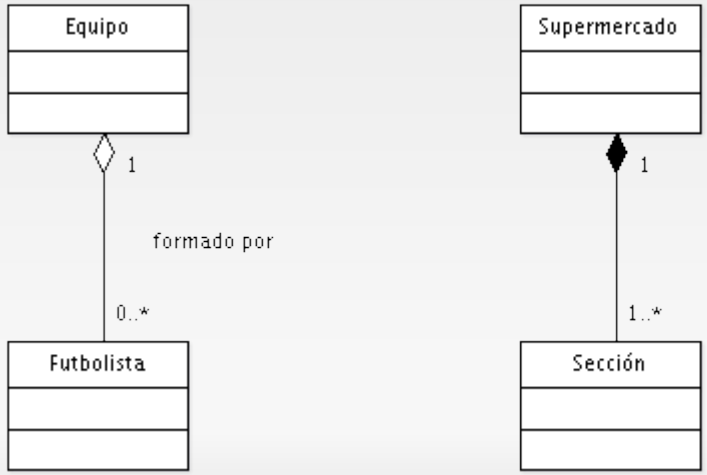


# Agregación y composición

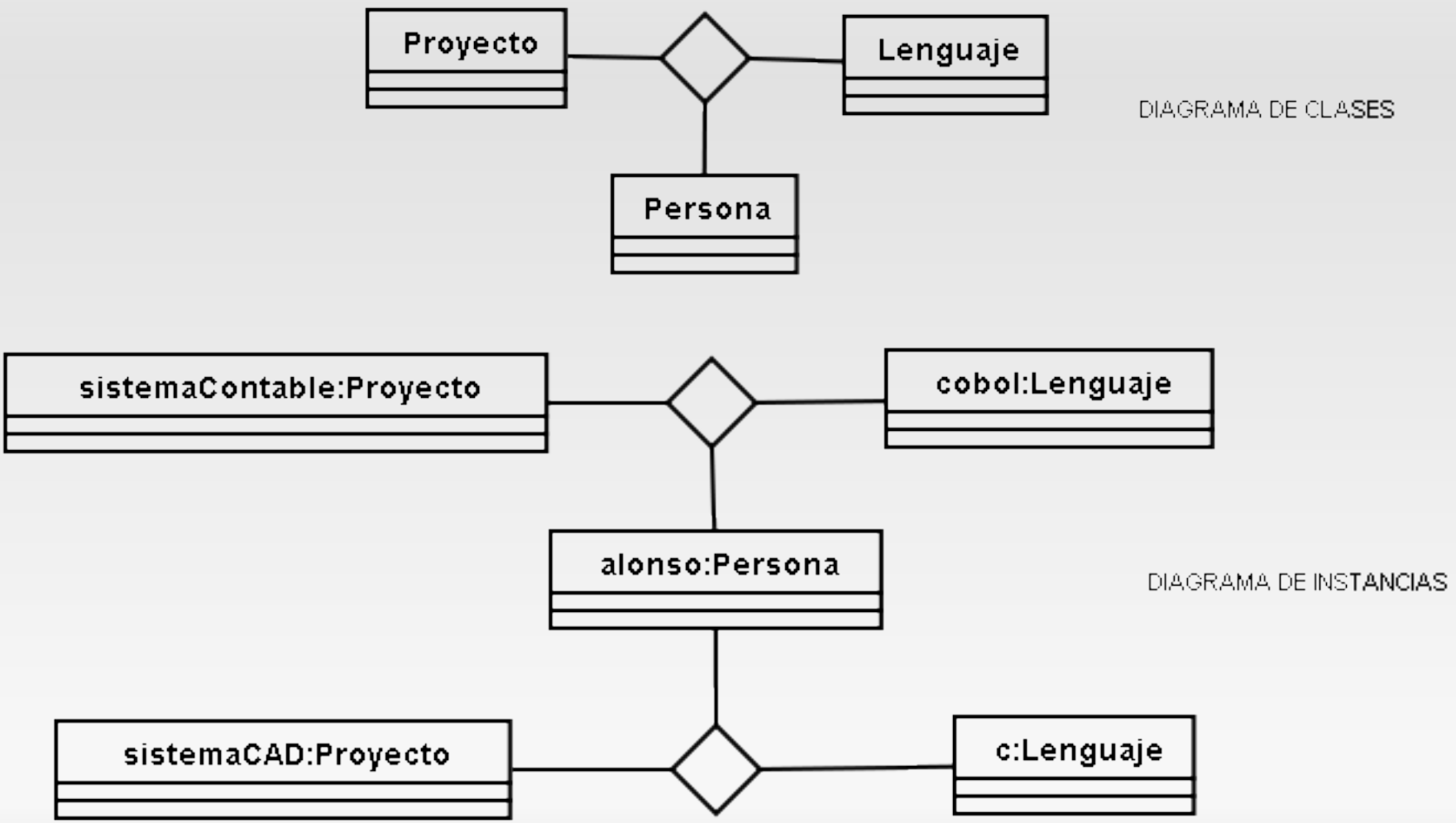
- La agregación es un tipo de relación jerárquica entre un objeto que representa la totalidad de ese objeto y las partes que lo componen.
- La composición es un tipo especial de agregación en la que los componentes dependen en existencia del objeto que representa el “todo”.
- La agregación se representa mediante un rombo blanco, y la composición con un rombo de color negro.

# Ejemplos de agregación y composición

- Si el supermercado dejara de existir, desaparecerían todas sus secciones.
- Si el equipo dejara de existir, no por ello desaparecerían los futbolistas que lo forman.



# Asociación ternaria



# Asociación Ternaria (II)

- En el ejemplo anterior, expresamos que una persona (programador) trabaja en un proyecto con un lenguaje de programación determinado.
- Si lo subdividimos en asociaciones binarias, perderemos información. Podremos determinar que una persona trabaja en varios proyectos y que conoce varios lenguajes, **pero no podremos determinar con qué lenguaje trabaja en cada proyecto**.

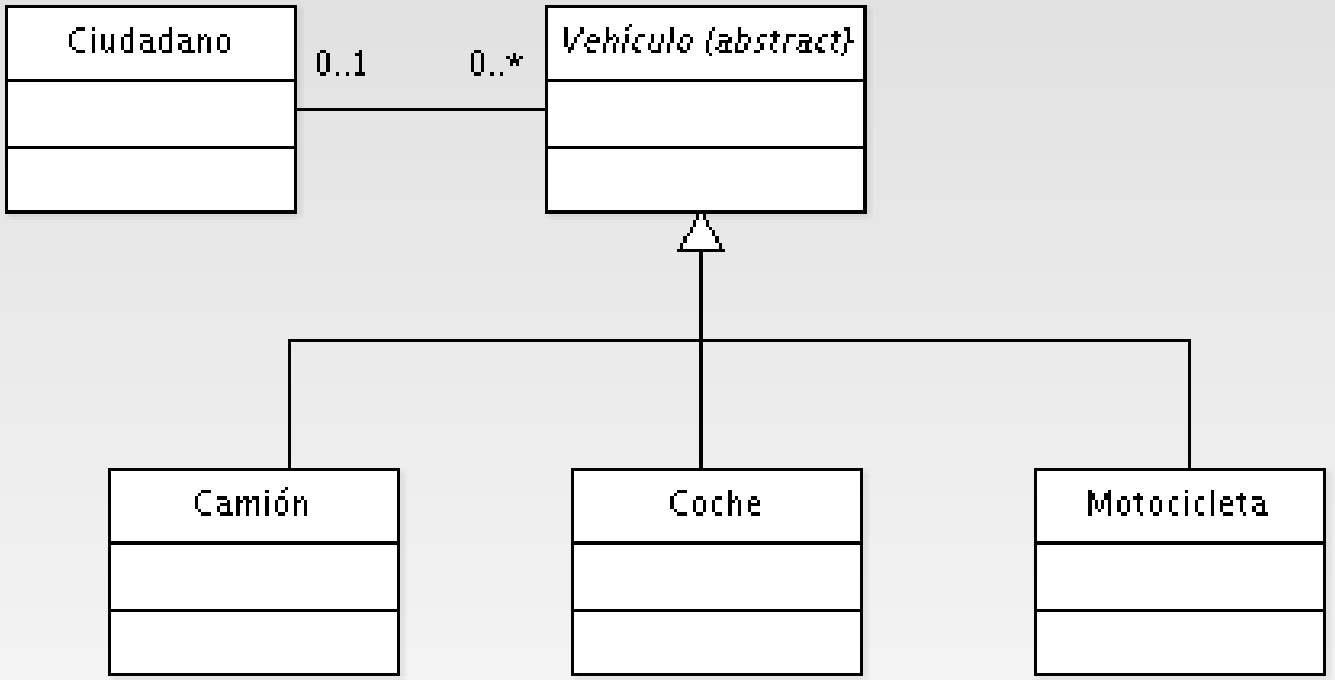
# Visibilidad

- Determina si un método o un atributo puede ser utilizado o accedido por otra clase. Hay cuatro niveles de visibilidad:
  - public (+)  Accesible por todas las clases
  - protected (#) Cualquier descendiente puede acceder a la característica
  - private (-) Sólo la propia clase puede acceder
  - package (~) Sólo las clases del propio paquete pueden acceder

# Elementos abstractos

- En las jerarquías de generalización, a veces deseamos expresar que una clase no puede tener instancias directas.
- A ese tipo de clases se las denomina **CLASES ABSTRACTAS.**
- Se suelen representar poniendo su nombre en cursiva o con la palabra reservada *{abstract}* junto al nombre

# Ejemplo clase abstracta

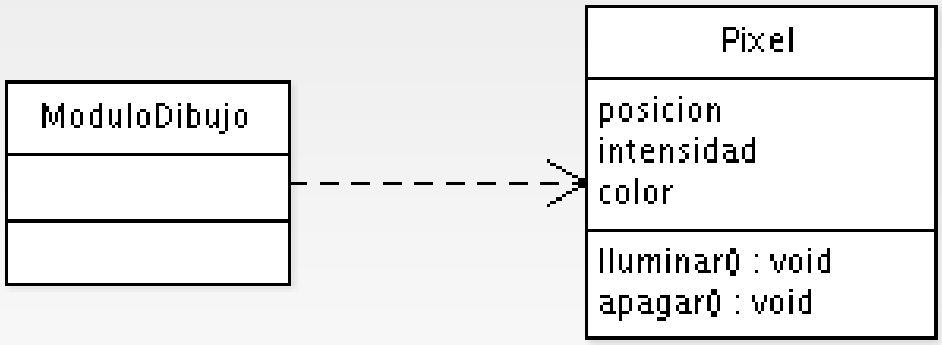


# Dependencia

- Es una relación de uso. Una clase utiliza la información y los servicios que proporciona otra clase, pero no necesariamente a la inversa.
- Un cambio en la especificación de un elemento, puede afectar a otro elemento que lo utiliza (que depende del anterior).
- Se representa con una flecha discontinua que apunta hacia el elemento del cual se depende.

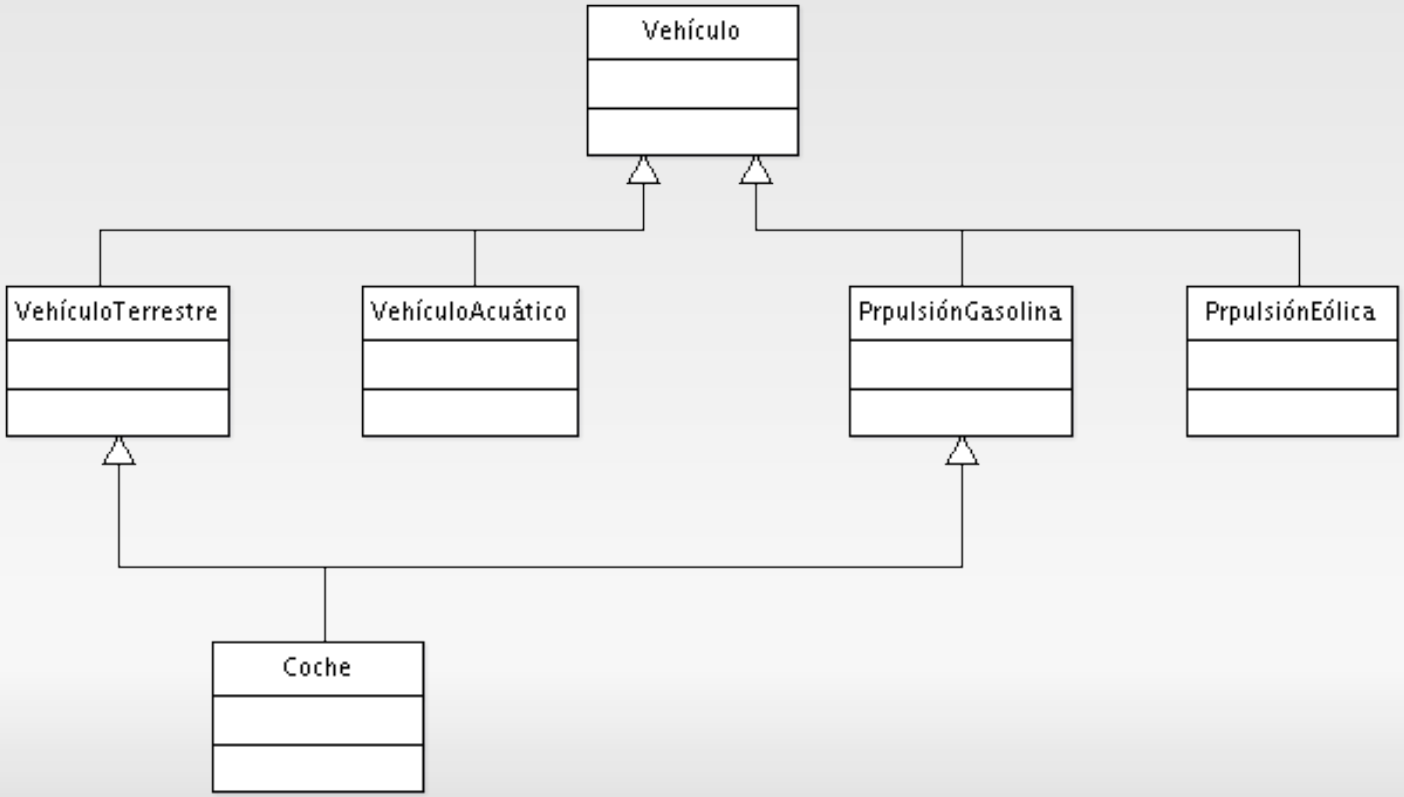
# Ejemplo de dependencia

- En el siguiente ejemplo se aprecia cómo la clase “ModuloDibujo” necesita usar la clase “Pixel” para llevar a cabo su cometido (permitir la realización de dibujos)



# Herencia múltiple

- Una clase hereda de más de un padre.
- No abusar de su uso.
- Algunos lenguajes, como por ejemplo java, no la implementan.

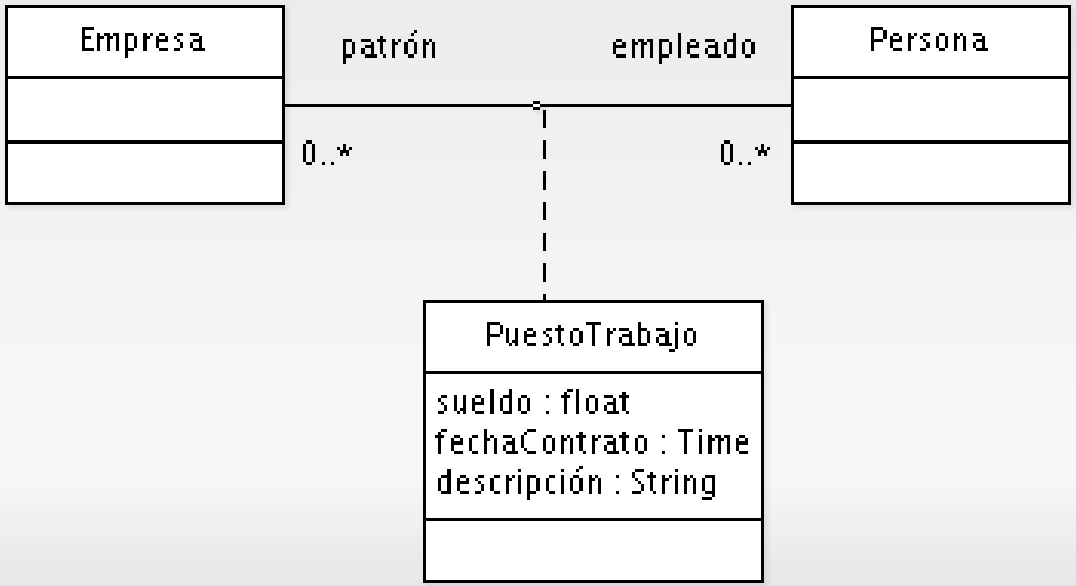


# Herencia múltiple: inconvenientes

- Herencia repetida: la clase que hereda múltiplemente (*coche* en el ejemplo) recibe los atributos de su *abuela* (Vehículo) tanto por parte de su *padre* (*VehículoTerrestre*) como por parte de su *madre* (*PropulsiónGasolina*).
- Ambigüedad: *VehículoTerrestre* y *PropulsiónGasolina* pueden tener métodos que se llaman igual pero hacen cosas diferentes... *coche hereda ambos.*

# Clases asociación

- Hay asociaciones que pueden tener propiedades (atributos y métodos)
- Las clases asociación tienen características tanto de asociación como de clase.



# Restricciones

- Sirven para matizar ciertos aspectos de las asociaciones.
  - *ordered*: El conjunto de objetos en un extremo de la asociación sigue un orden.
  - *set:* Objetos únicos, sin duplicados.
  - *bag:* Objetos no únicos, puede haber duplicados.
  - *ordered set*: Objetos únicos pero ordenados.
  - *list* o *sequence:* Objetos ordenados, pudiendo haber duplicados.
  - *readonly*: Un enlace, una vez añadido desde un objeto del otro extremo de la asociación, no se puede modificar ni eliminar.