

## Lección 2.2

Diagramas estructurales para  
la representación de clases

# Objetivos de aprendizaje



- Conocer y comprender el significado de los distintos elementos gráficos de un **diagrama de clases** de UML para el diseño de sistemas software.
- Conocer y comprender el significado de los distintos elementos gráficos de un **diagrama de paquetes** de UML.
- **Implementar** una estructura de clases a partir de unos diagramas de clases y de paquetes dados.
- Aprender a **modelar** sistemas simples.

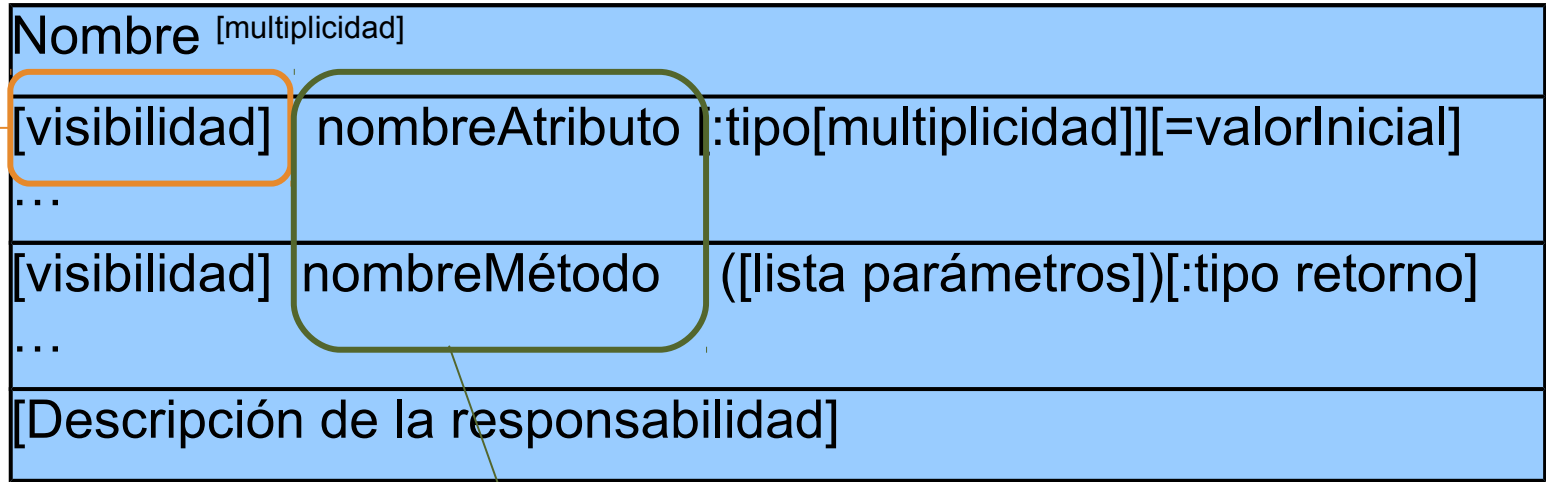
# Contenidos



1. **Diagrama de clases de UML:** elementos, significado y representación gráfica.
2. **Generación de código** desde un diagrama de clases del diseño UML.
3. **Diagrama de paquetes en UML:** elementos, significado, representación gráfica y generación de código.
4. **Modelando** diagramas de clases.

# 1. Diagrama de clases UML

## Clases:



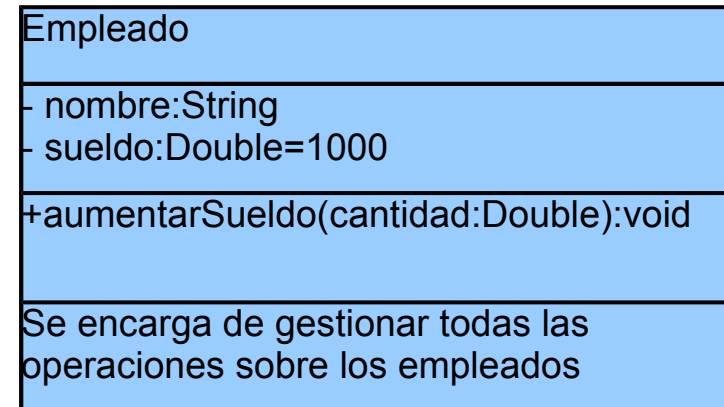
+ pública  
- privada  
~ paquete  
# protegida

¿Recuerdas qué significan?  
(lección 2.1)

Por defecto, son  
atributos o métodos  
de instancia.  
Los atributos y  
métodos de clase se  
indican subrayando  
el nombre.

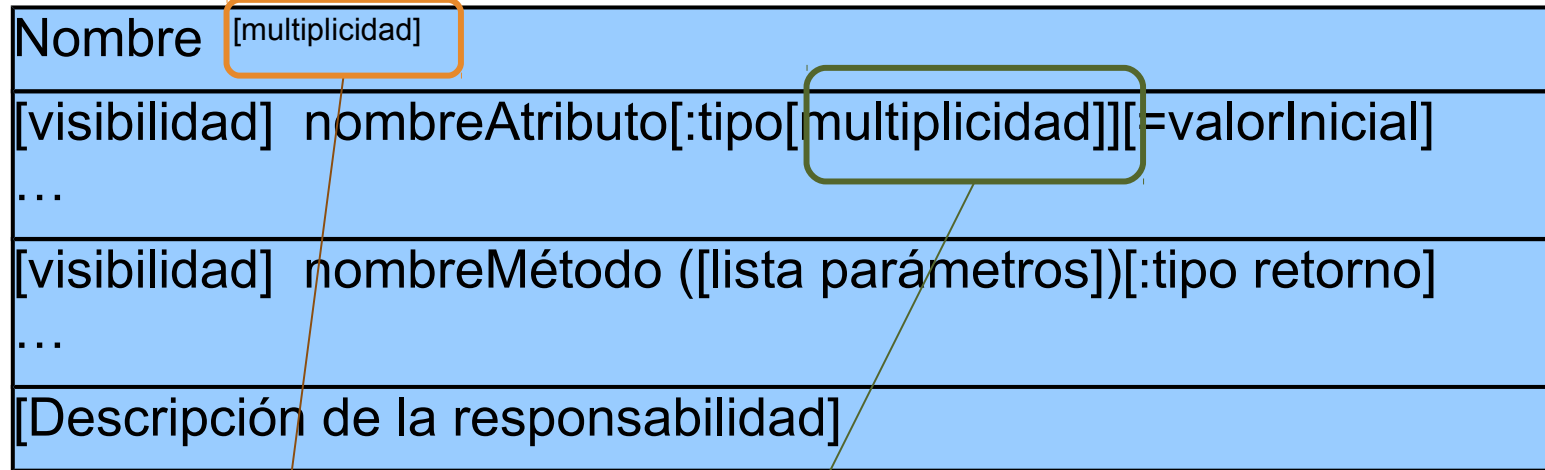
¿Recuerdas la diferencia?  
(lección 2.1)

## Ejemplo:



# 1. Diagrama de clases UML

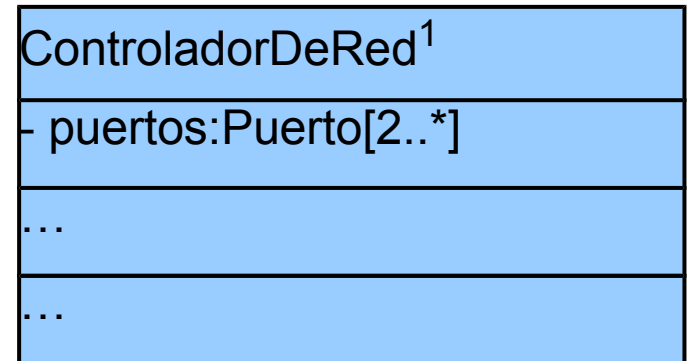
## Clases:



*Multiplicidad de clase:*  
número de instancias que  
puede tener una clase.

*Multiplicidad de  
atributo:* número de  
elementos en dicho  
atributo.



## Ejemplo



# 1. Diagrama de clases UML

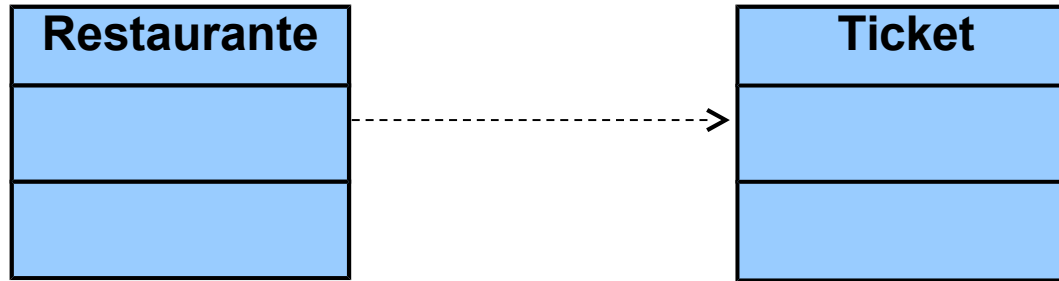
## Relaciones entre clases:

Existen fundamentalmente cuatro tipos de relaciones:

- Dependencia
  - Asociación
- 
- A continuación
- Realización
  - Generalización
- 
- En el tema 3

# 1. Diagrama de clases UML

- **Relación de dependencia:** modela una relación débil y poco duradera en el tiempo.

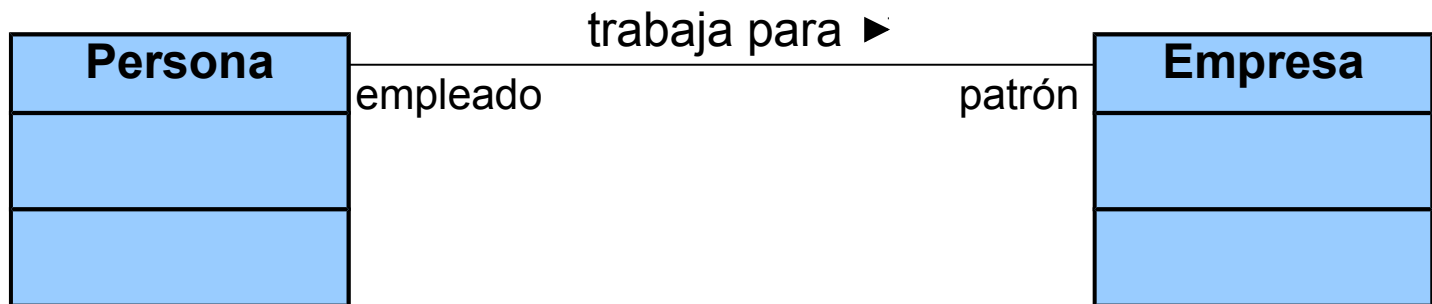


- **Relación de asociación:** modela una relación estructural fuerte y duradera en el tiempo.



# 1. Diagrama de clases UML

- Elementos de las asociaciones:
  - **Nombre:** Nombre de la asociación. Se puede indicar hacia dónde se lee empleando ►
  - **Rol:** Se emplea para explicitar el papel que juega cada clase en la asociación

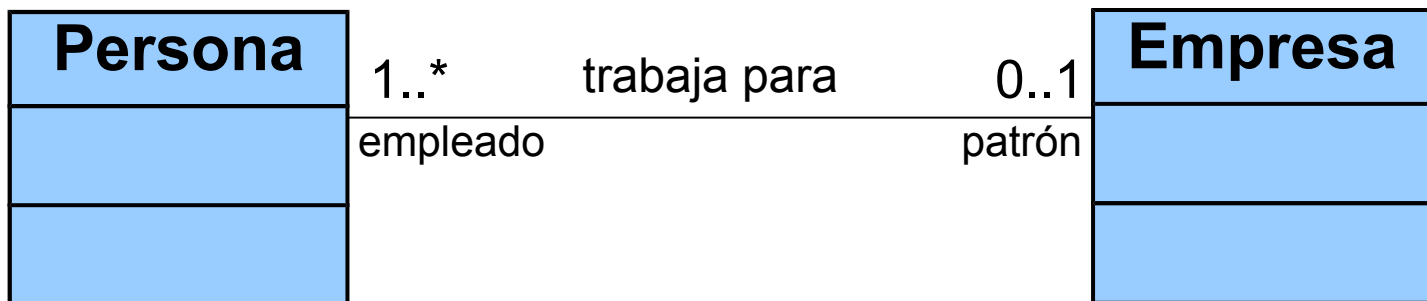


Ojo: No se debe confundir la dirección de lectura del nombre de la asociación con la navegabilidad de la misma (transparencia siguiente).





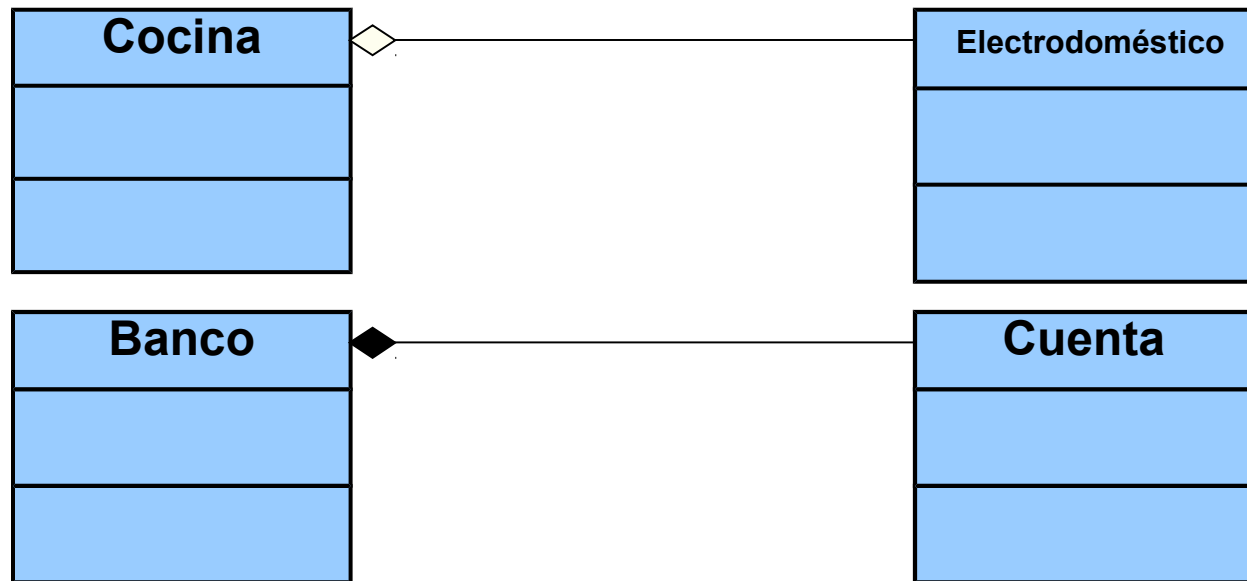
# 1. Diagrama de clases UML

- **Elementos de las asociaciones:**
  - **Navegabilidad:** Representa si el conocimiento entre los objetos de los extremos de la asociación es unidireccional ( $\rightarrow$  o  $\leftarrow$ ) o bidireccional (sin punta de flecha en ninguno de los extremos).
  - **Multiplicidad:** Indica el número de objetos de una clase que pueden asociarse con un objeto de la otra clase.
    - Su sintaxis es *valorMínimo..valorMáximo*, pudiendo usarse \* para especificar “varios”, y así no dar un valor máximo concreto.
    - Si no se indica nada, por defecto es 1.



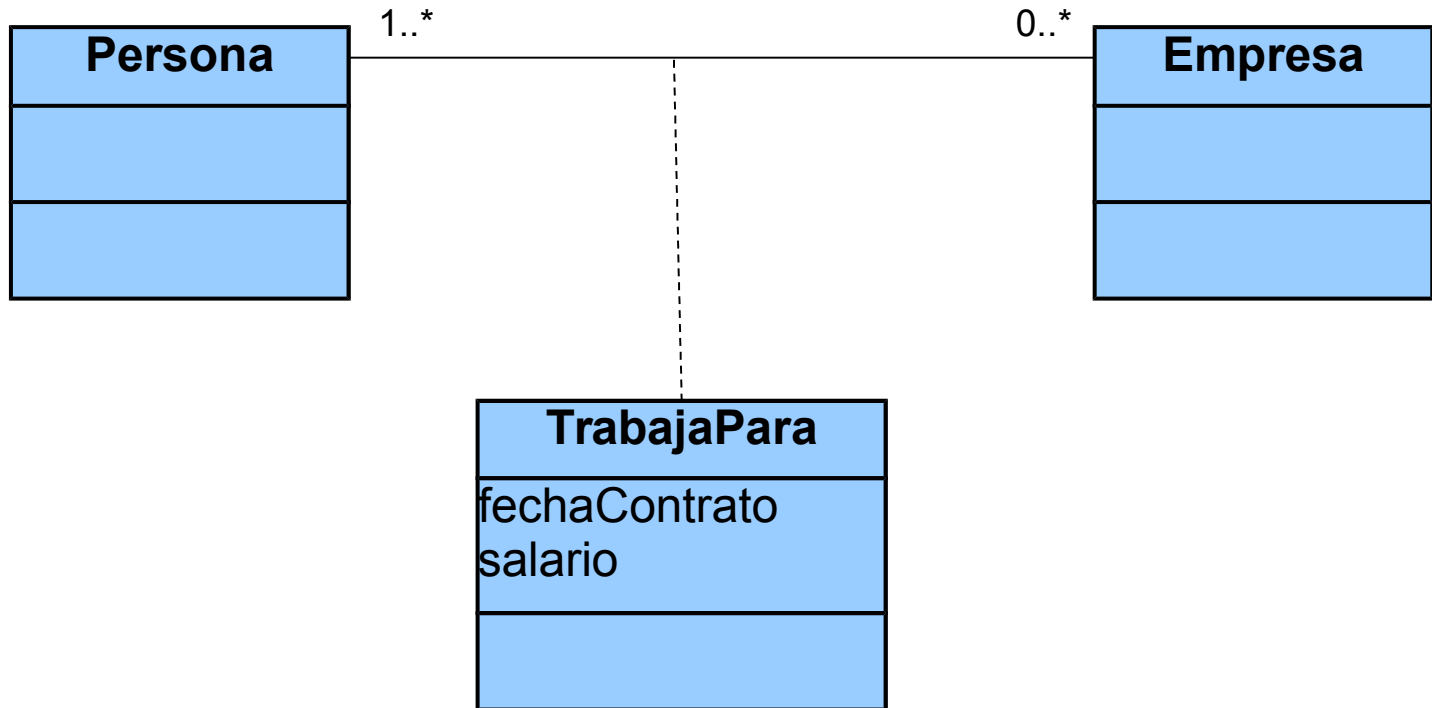
# 1. Diagrama de clases UML

- Asociaciones especiales:
  - **Agregación** (  ): Asociación cuyo nombre es “parte de”, en la que una de las clases representa el “todo” y la otra la/s “parte/s”.
  - **Composición** (  ): Se trata de una agregación fuerte, en que la/s parte/s no tiene/n sentido sin el todo.



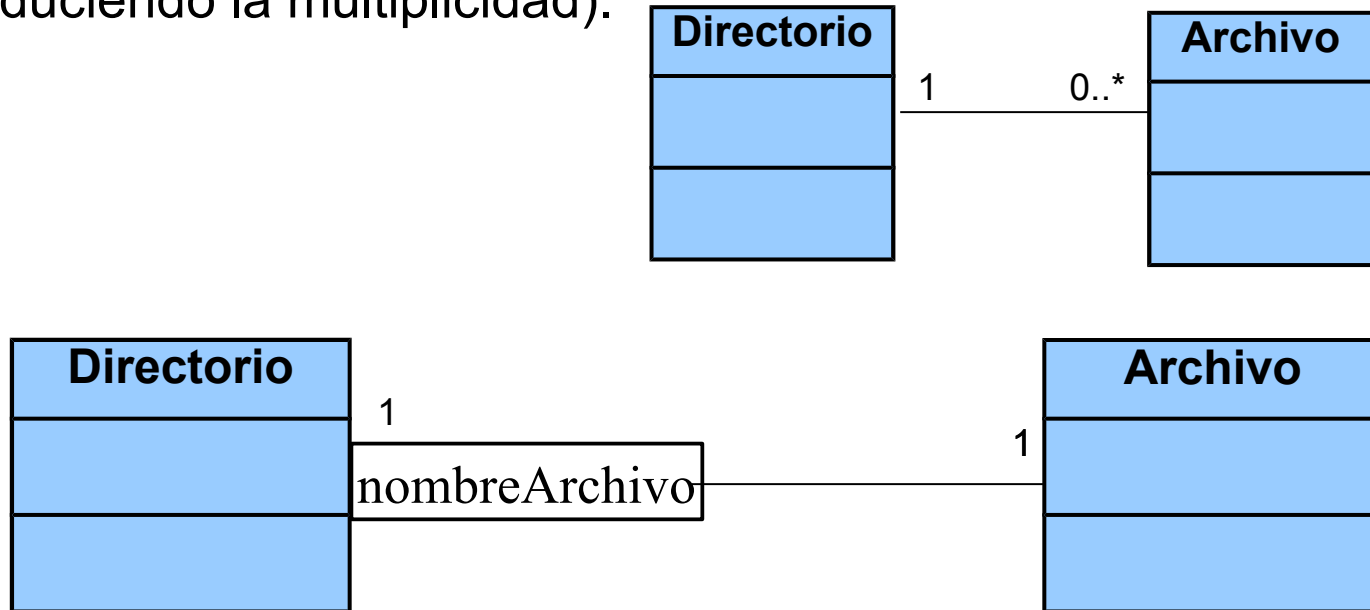
# 1. Diagrama de clases UML

- Clase asociación:
  - Se da cuando una asociación presenta atributos propios y tiene que ser modelada como una clase.



# 1. Diagrama de clases UML

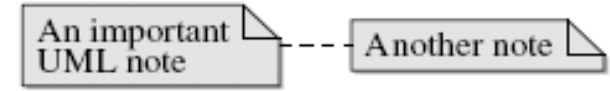
- Cualificadores de la asociación:** un **cualificador** es un atributo de algunas de las clases de la asociación que pasa a ser un atributo asociado a la clase del otro extremo (normalmente reduciendo la multiplicidad).



- Se suelen emplear colecciones para modelar las asociaciones cualificadas, empleando el cualificador como identificador. Así, por ejemplo, *nombreArchivo* es un atributo de la clase *Archivo*, que se emplea en *Directorio* como identificador único de cada uno de los archivos del directorio.

# 1. Diagrama de clases UML

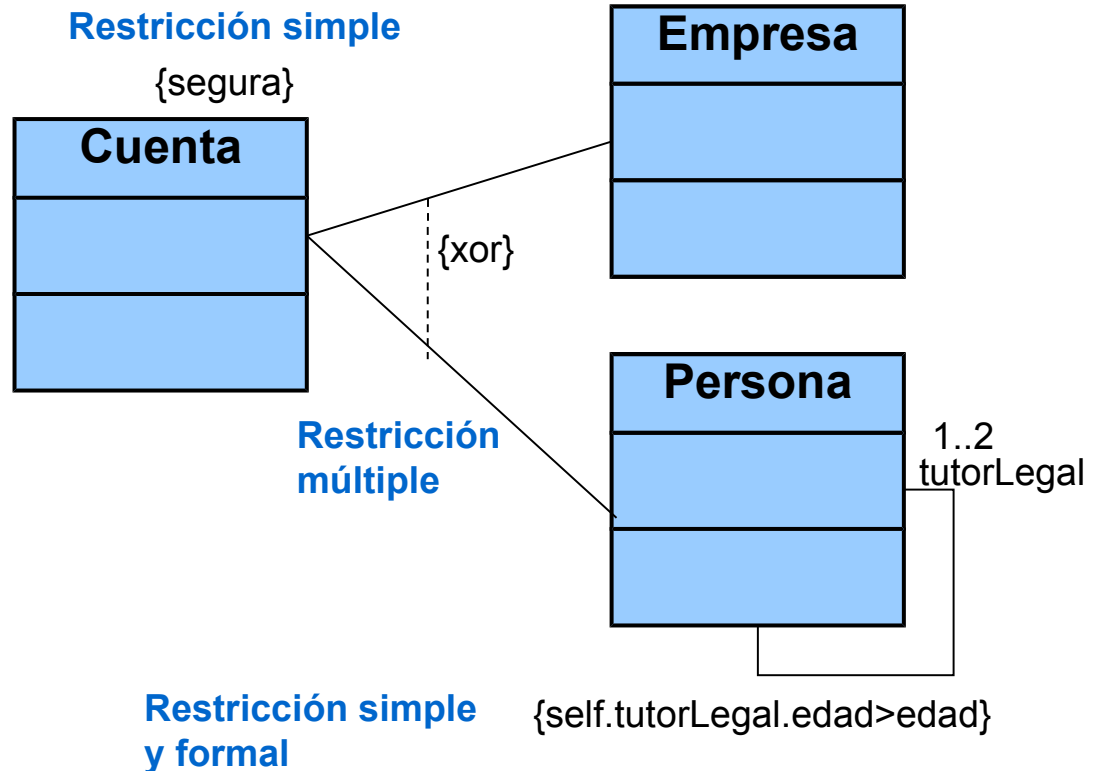
**Notas:** Permiten añadir comentarios en lenguaje natural.



## Restricciones:

Son extensiones de la semántica de los elementos del lenguaje, añadiendo nuevas reglas o modificando las ya existentes.

Se representan entre llaves.



## 2. Generación de código

A partir de un diagrama de clases es posible codificar la estructura de las clases como sigue:

| Paso | Elementos del diagrama de clases  | Traducción en la implementación                                  |
|------|-----------------------------------|--|
| 1    | Clases (con atributos y métodos). | Clases (con declaración de atributos y cabecera de los métodos). |
| 2    | Asociaciones entre clases.        | Atributos de referencia.   |

- Las relaciones de dependencia se implementan en la cabecera y en el cuerpo de los métodos.
- Para completar el código tan sólo faltaría la implementación de los métodos y añadir la herencia (lo veremos en las lecciones siguientes).

## 2. Generación de código

### Paso 1: Definición de las clases

Para cada clase del diagrama, se codifica:

1. Una clase con el mismo nombre.
2. Tantos atributos como tenga la clase, con la visibilidad, nombre y tipo indicados en el diagrama.
3. La interfaz (cabecera) de los métodos, con las mismas opciones de visibilidad, tipo de valor devuelto, nombre y misma lista de atributos (con idénticos nombres y tipos) que aparecen en el diagrama.

## 2. Generación de código



```
class A

=begin
Recuerda que los atributos de
clase se establecen con @@ y los
de instancia con @ en cualquier
método de instancia o en
attr_reader, attr_writer o
attr_accessor
=end
  @@b  # ámbito de clase
  def initialize(unC)
    @c=unC
  end
  def self.opel
    #Aquí irá el código
  end
  def ope2(arg1)
    #Aquí irá el código
  end
end
```

| A  |
|--|
| - <u>b</u> : B<br>- c:C                          |
| + <u>ope1</u> ():void<br>+ ope2(arg1:float): int |



```
class A {

/*Recuerda que las variables de
clase se definen con static*/

    private static B b;
    private C c;
    public static void opel(){
        //Aquí irá el código
    }
    public int ope2(float arg1){
        //Aquí irá el código
    }
}
```



## 2. Generación de código

### Paso 2: Definición de los atributos de referencia

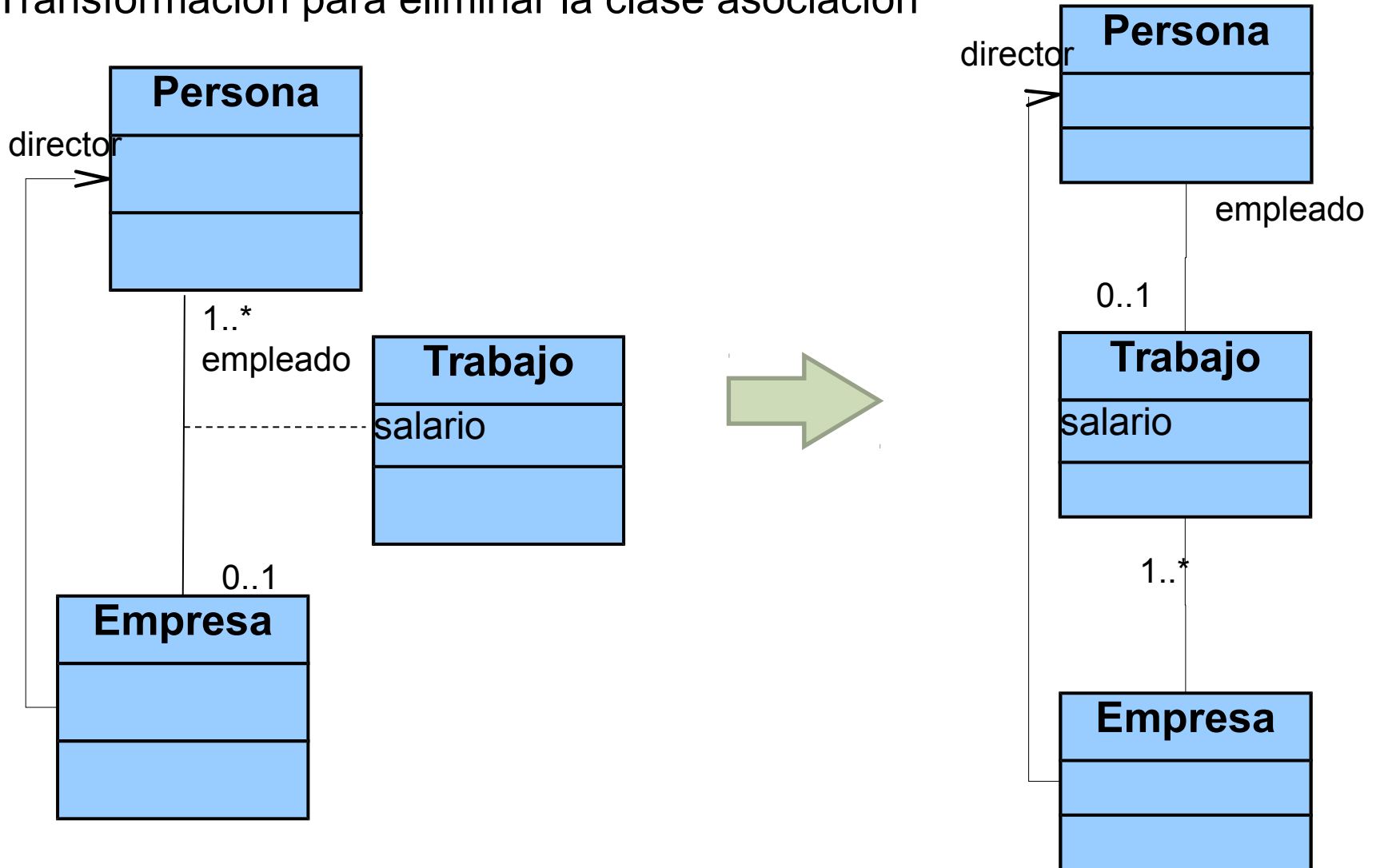
Los **atributos de referencia** son aquellos que surgen a raíz de las **asociaciones** entre las clases. Para identificarlos, hay que prestar atención a la navegabilidad, multiplicidad, rol y atributos de la asociación.

- Transformar el diagrama de clases en el que haya clases asociación, en otro que no las contenga.
- Si la **navegabilidad** es unidireccional, se incluye un atributo de referencia en el origen de la navegabilidad. Si es bidireccional, en ambos extremos.
- Si se conoce el nombre de **rol** de los objetos, se debe utilizar como nombre de los atributos de referencia.
- Si en el destino de la navegabilidad la **multiplicidad** es distinta de 1, el atributo de referencia debe ser de tipo clase contenedora (o colección) de los objetos de esa clase.

## 2. Generación de código

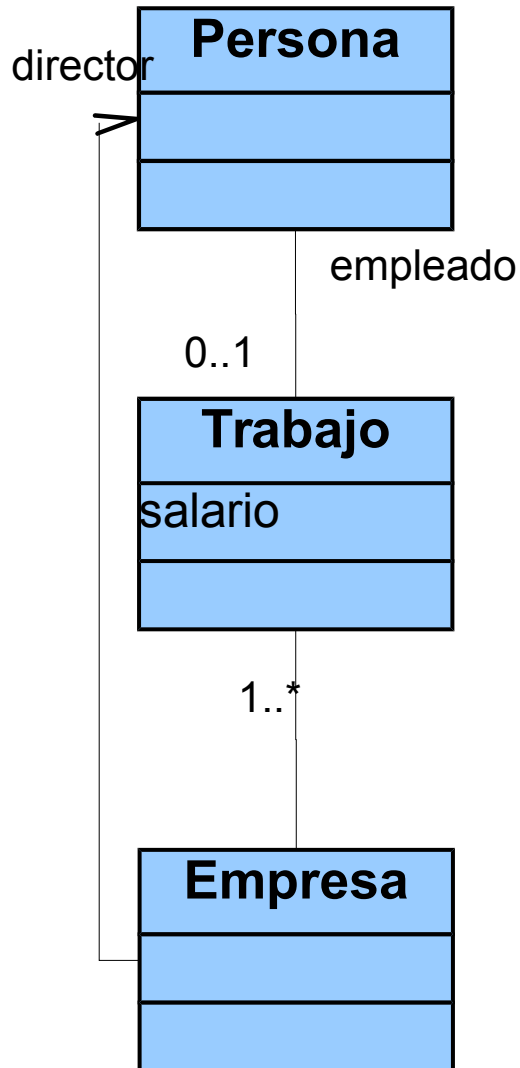
### Paso 2: Definición de los atributos de referencia

Transformación para eliminar la clase asociación



## 2. Generación de código

### Paso 2: Definición de los atributos de referencia



```
class Persona {
    private Trabajo miTrabajo;
}
```


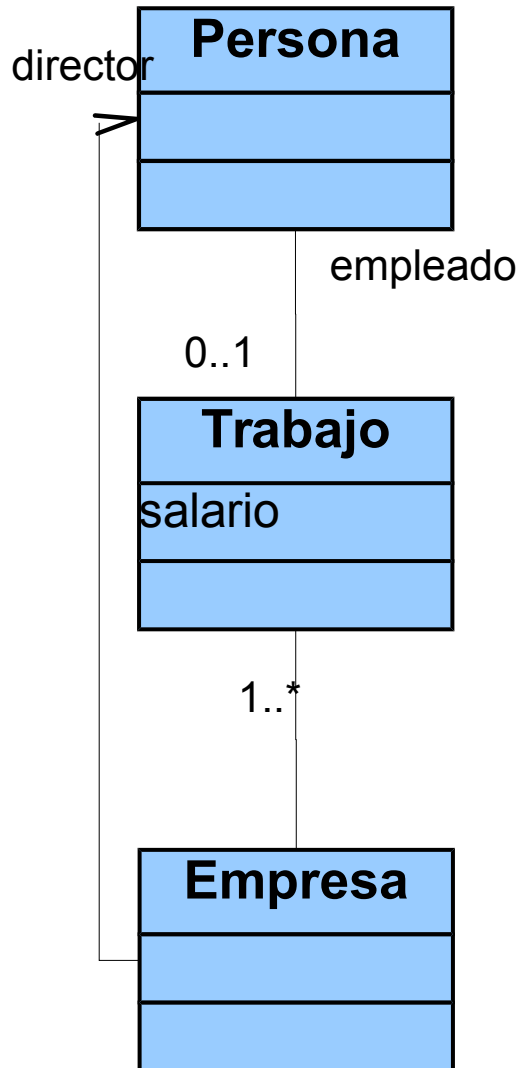
```
class Empresa {
    private ArrayList<Trabajo>
    trabajos;
    private Persona director;
}
```

¿Cuáles son los atributos de referencia de la clase Trabajo? Genera el código en Java



## 2. Generación de código


### Paso 2: Definición de los atributos de referencia



```
class Persona
  def initialize(trabajo)
    @miTrabajo = trabajo
  end
end
```

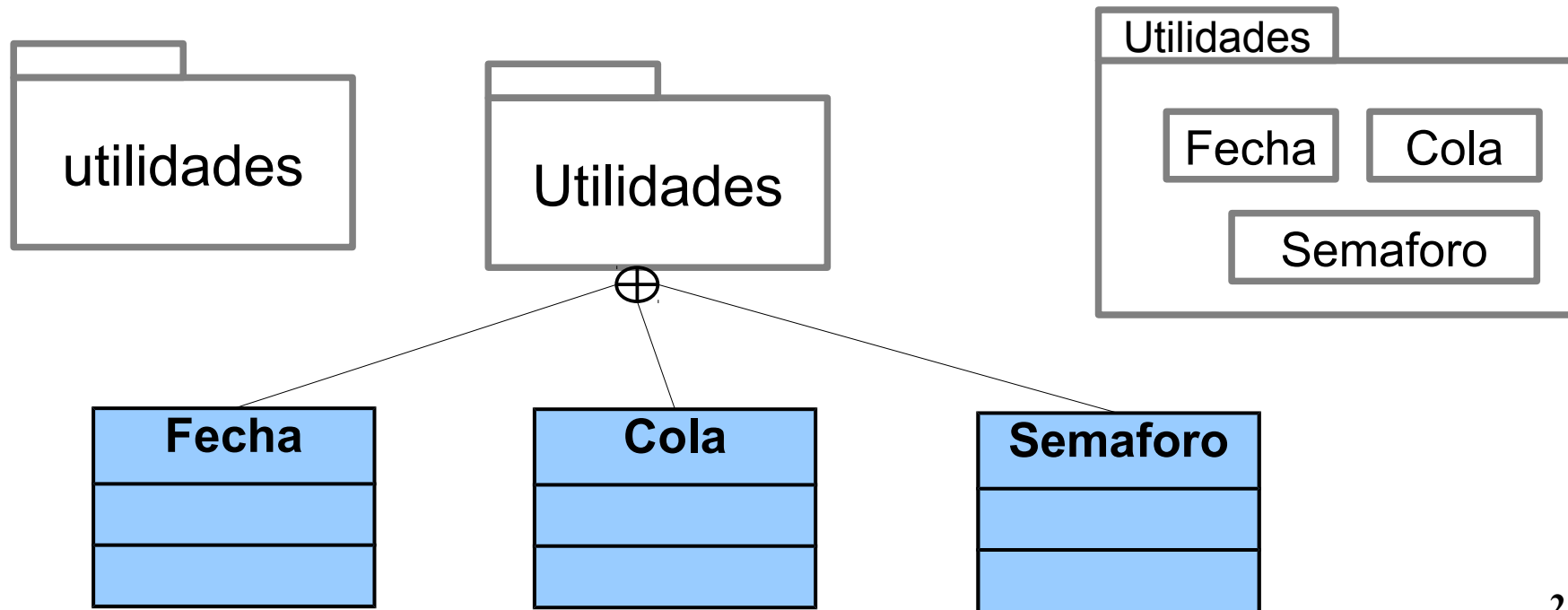
```
class Empresa
  def initialize(director)
    @trabajos=Array.new
    @director = director
  end
end
```

¿Cuáles son los atributos de referencia de la clase Trabajo? Genera el código en Ruby



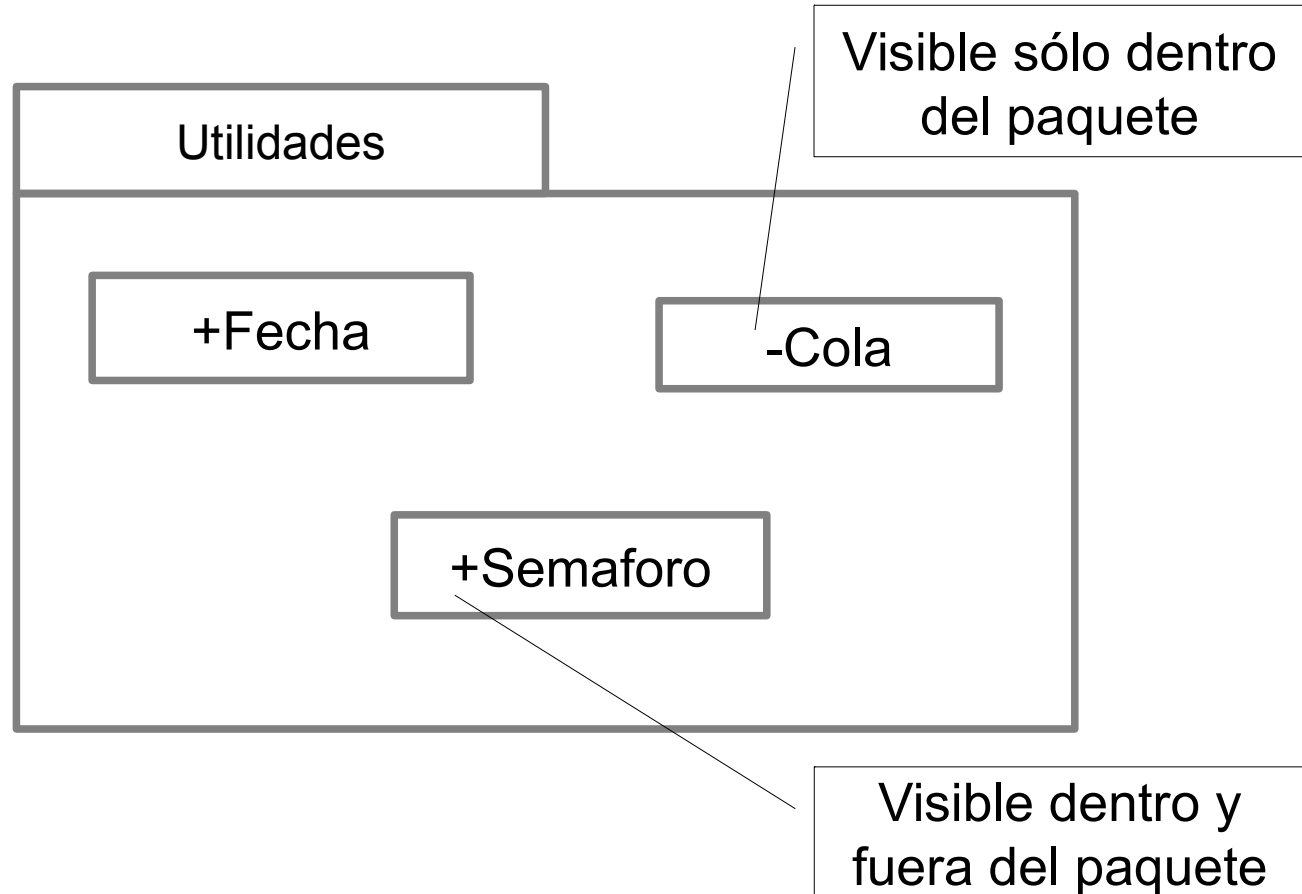
# 3. Diagrama de paquetes UML

- Un paquete permite representar agrupaciones de clases
- Los diagramas de paquetes están compuestos por los paquetes y sus relaciones (dependencia).
- Un paquete puede estar contenido a su vez en otro paquete (paquetes anidados).
- Existen distintas representaciones:



# 3. Diagrama de paquetes UML

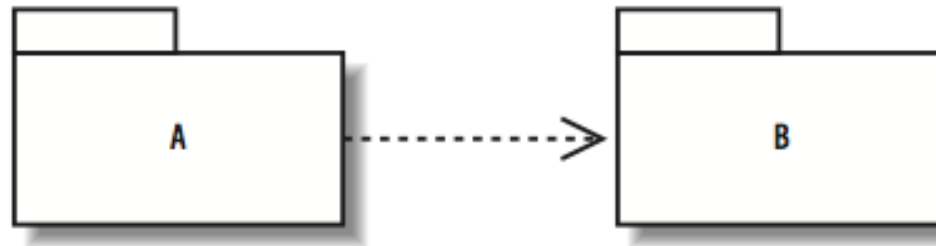
- **Visibilidad** de los elementos de un paquete



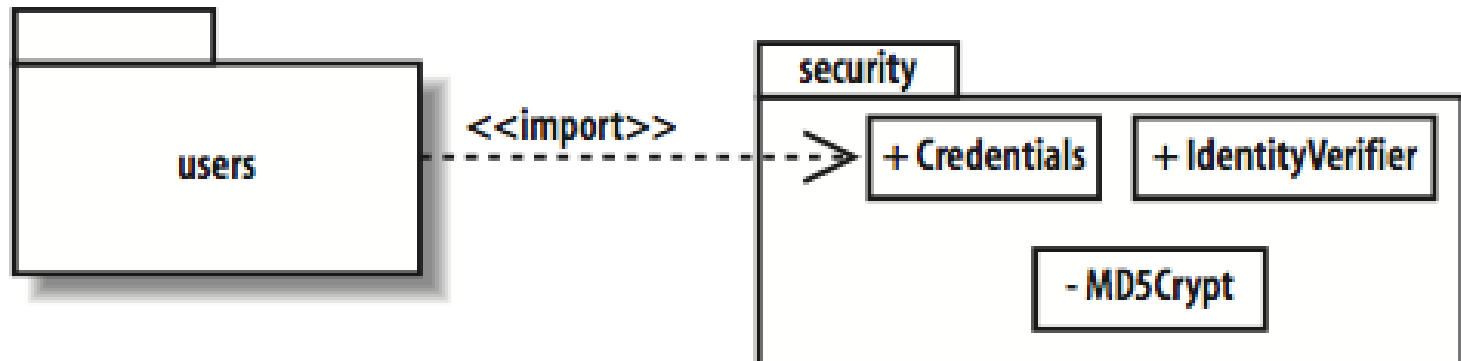
# 3. Diagrama de paquetes UML

- **Relaciones entre paquetes:** las únicas relaciones que se dan entre paquetes son las dependencias. Éstas se pueden etiquetar para indicar el tipo de dependencia.

**Ejemplo:** El paquete A **usa** algún elemento del paquete B.



**Ejemplo:** El paquete *users* **importa** la clase *Credentials* del paquete *security*



# 3. Diagrama de paquetes UML

- Los diagramas de paquetes permiten detectar dependencias complejas y ciclos que rompen el principio de **bajo acoplamiento**.

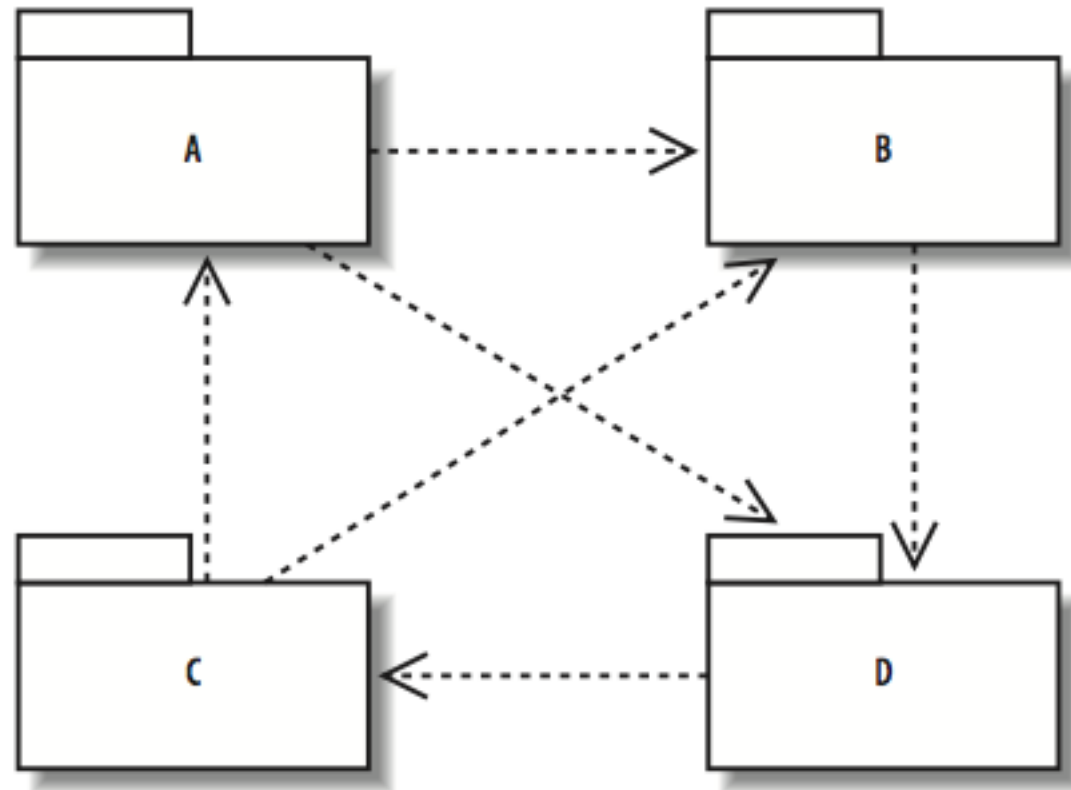


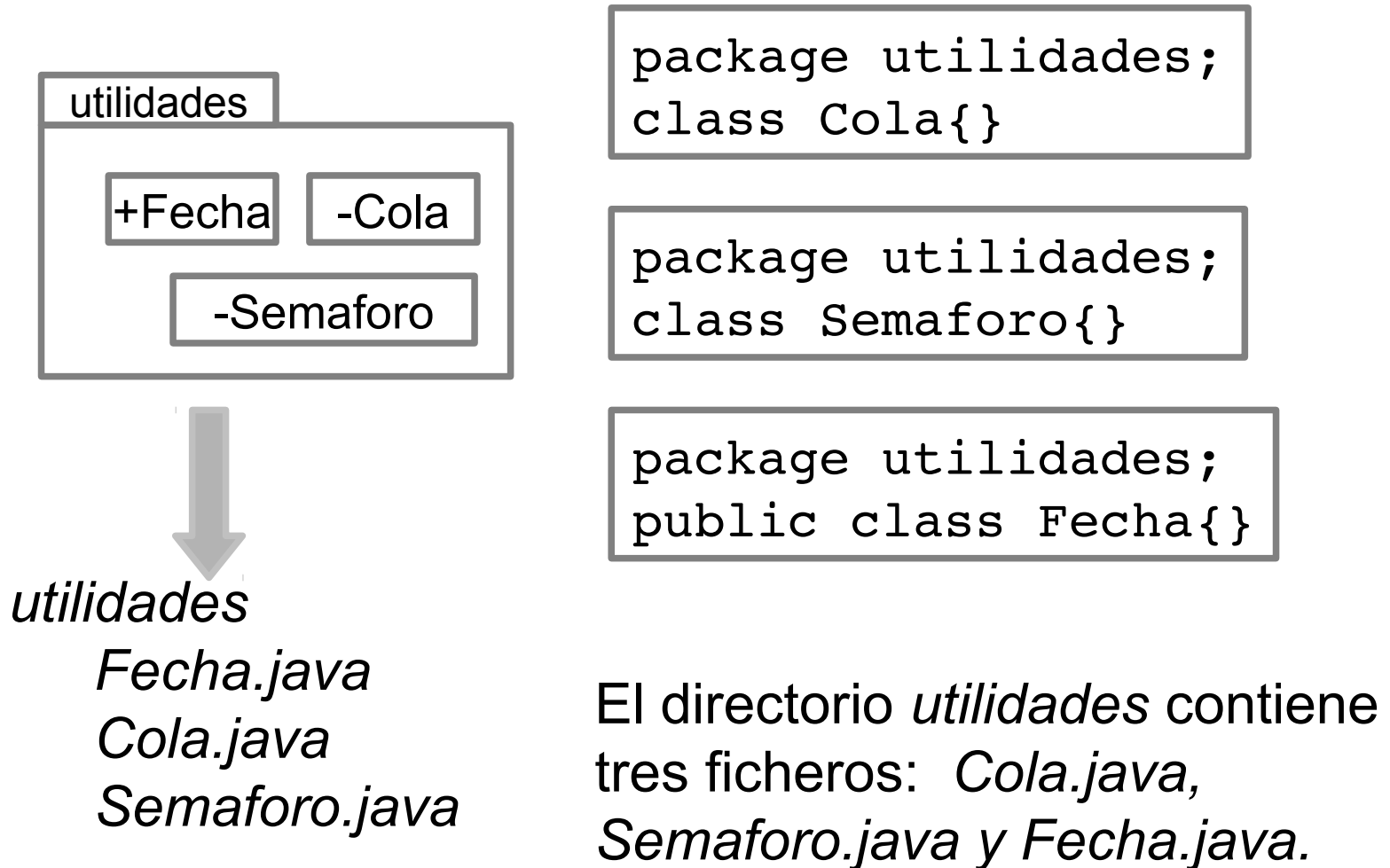
Figura del libro “Learning UML 2.0”, O'Reilly



# 3. Diagrama de paquetes UML



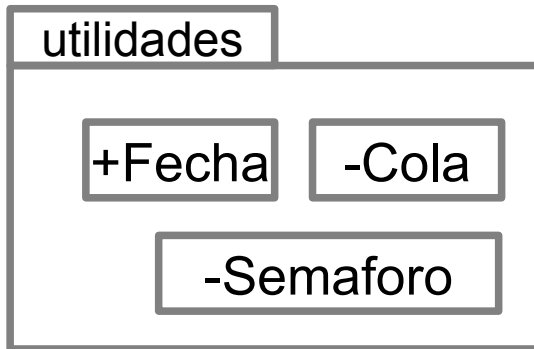
- **Implementación en Java** del paquete, correspondencia con directorios y archivos



# 3. Diagrama de paquetes UML



**Implementación en Ruby** del paquete mediante la estructura `module`.



```
module Utilidades
  class Cola
    ...
  end
  private_constant :Cola

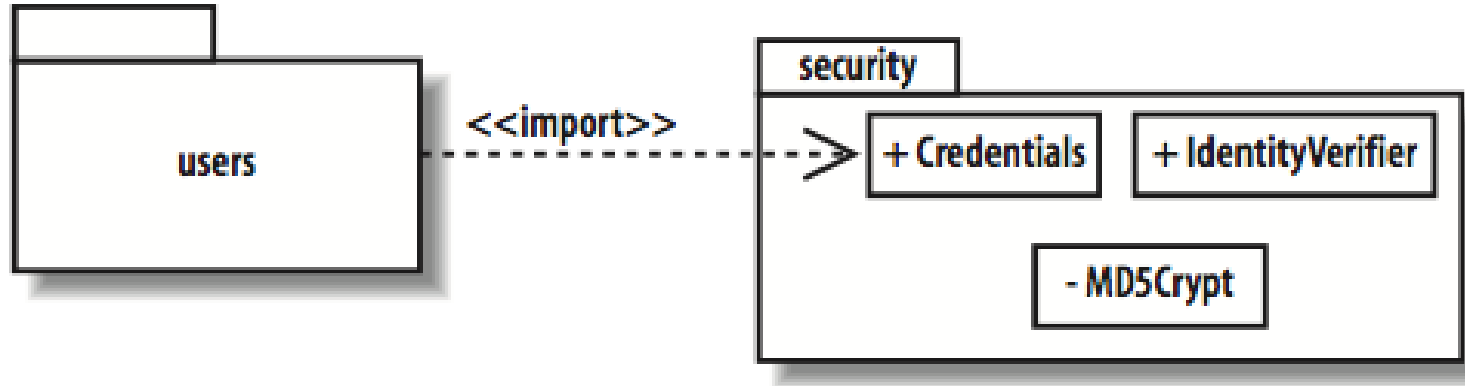
  class Semaforo
    ...
  end
  private_constant :Semaforo

  class Fecha
    ...
  end
end
```

# 3. Diagrama de paquetes UML



Implementación en Java de las relaciones entre paquetes.



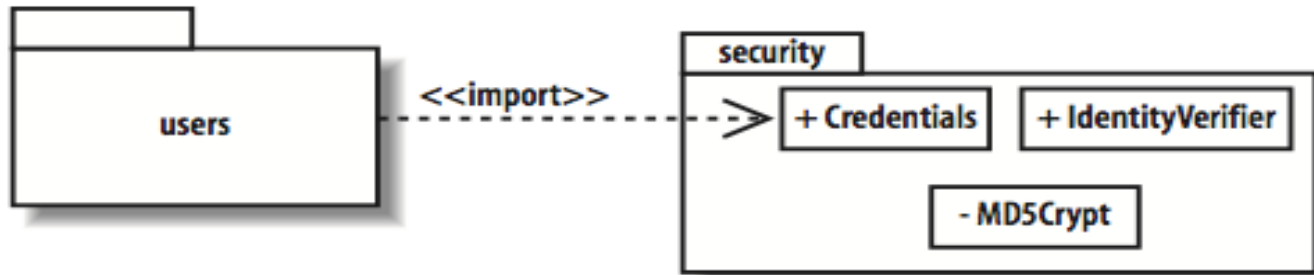
Formas de acceder a *una clase que se importa*:

`import security.Credentials;` // antes de la cabecera de la clase de users que necesite de *Credentials*

`security.Credentials` // en cualquier línea de código de una clase de users, seguido del método o variable a los que se desee acceder, siempre que sean públicas

# 3. Diagrama de paquetes UML

## Implementación en Ruby de las relaciones entre paquetes.



1. En el fichero con clase de users que quiera usar *Credentials*, añadir:
  - *require 'security'* ,o
  - *require\_relative 'security'* # si están en el mismo proyecto y al mismo nivel  
(Donde *security* es el nombre de archivo con el módulo que define la clase *Credentials*)
2. Después, para poder “usar” la clase *Credentials*, dentro de una clase de users se puede escribir lo siguiente:
  - *include Security* # copia todo el código del módulo *Security* en el lugar donde se incluya esta línea (ojo!!!)
  - *Security::Credentials* # referencia a la clase *Credentials* cada vez que quiera usarse

## 4. Modelando Diagramas de clases

El **modelado de un Diagrama de Clases** es una tarea compleja, sobre todo cuando el problema a modelar es complejo, en estos casos hay que seguir alguna **metodología de Diseño**.

Para sistemas simples, partiendo de una breve descripción del problema, podemos seguir el siguiente procedimiento:

1. **Identificar las clases:** Conceptos relevantes del problema (sustantivos)
2. **Identificar los atributos** de las clases: Propiedades asociadas a conceptos (sustantivos)
3. **Identificar las operaciones:** Responsabilidad de las clases (verbos)
4. **Identificar las asociaciones:** Relaciones relevantes entre conceptos (frases que relacionen conceptos)
5. **Refinar el diagrama de clases** obtenido hasta obtener un diagrama que presente un buen diseño

Ejemplo en las siguientes transparencias

# 4. Modelando Diagramas de clases

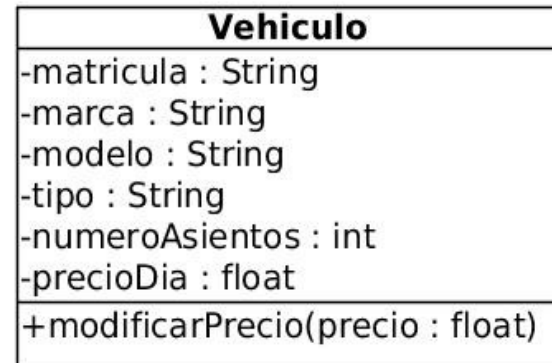
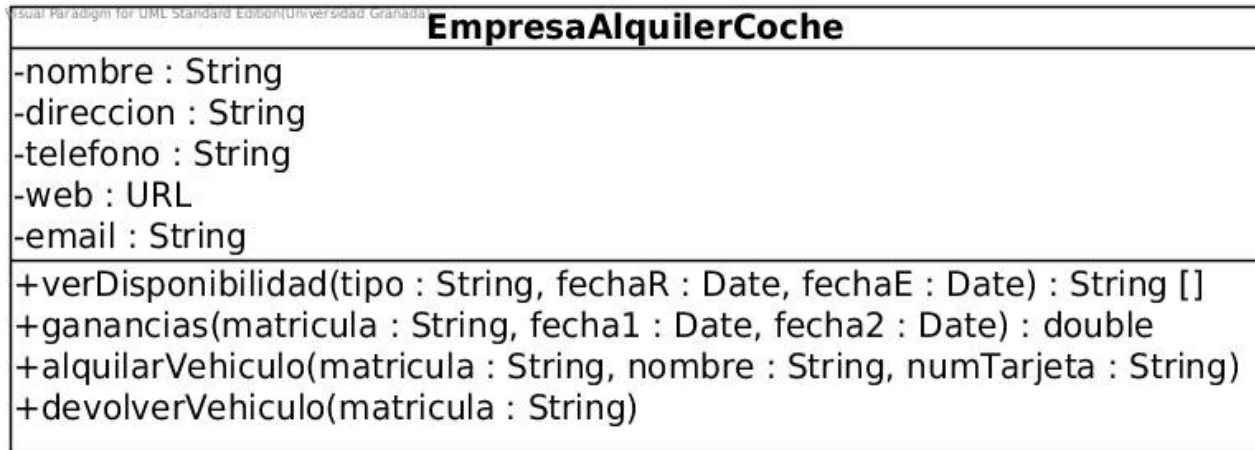
## 1. Identificando conceptos, 2. atributos y 3. operaciones

Una empresa de alquiler de vehículos posee una amplia flota de vehículos. La información relevante de cada vehículo es: matrícula, marca, modelo, el número de plazas y el tipo de vehículo. Cuando un cliente solicita un vehículo para su alquiler debe indicar la fecha de recogida, la fecha de devolución, el número de plazas necesarias, el tipo de vehículo y el número de tarjeta de crédito con que se abonará el alquiler. La empresa **comprueba la disponibilidad** en ese momento de algún vehículo del tipo solicitado, si lo hay, se le alquila al cliente, debiéndose registrar la fecha de recogida, la fecha de devolución, el importe del alquiler, **calculado** a partir de precio por día del vehículo y los días que va a estar alquilado. En caso de no tener un vehículo disponible, se lo comunica al cliente y se anula la solicitud. La empresa desea conocer los **vehículos alquilados en un determinado momento** y un informe con las **ganancias generadas por un determinado vehículo** en un periodo de tiempo. Cuando un cliente devuelve el vehículo desaparece su información del sistema.

# 4. Modelando Diagramas de clases

1. **Conceptos relevantes:** Empresa de alquiler de vehículos, Vehículo, Cliente y Alquiler.
2. **Atributos:**
  - Empresa de alquiler de vehículos:** nombre, dirección, teléfono, web (los deducimos del contexto)
  - Vehículo:** matrícula, marca, modelo, el número de plazas, tipo, precio de alquiler por día.
  - Cliente:** Nombre, numero de tarjeta.
  - Alquiler:** fecha de recogida, la fecha de devolución, importe.
3. **Operaciones o responsabilidades:**
  - Empresa de alquiler de vehículos:** ver disponibilidad de vehículos de un determinado tipo para unas determinadas fechas, proporcionar las ganancias de un determinado vehículo, en un periodo de tiempo, alquilar vehículo, devolver vehículo.
  - Vehículo:** Consultores de sus atributos y modificador de precio del día
  - Cliente:** Consultores de sus atributos.
  - Alquiler:** Consultores de atributos, cálculo total del alquiler.

# 4. Modelando Diagramas de clases



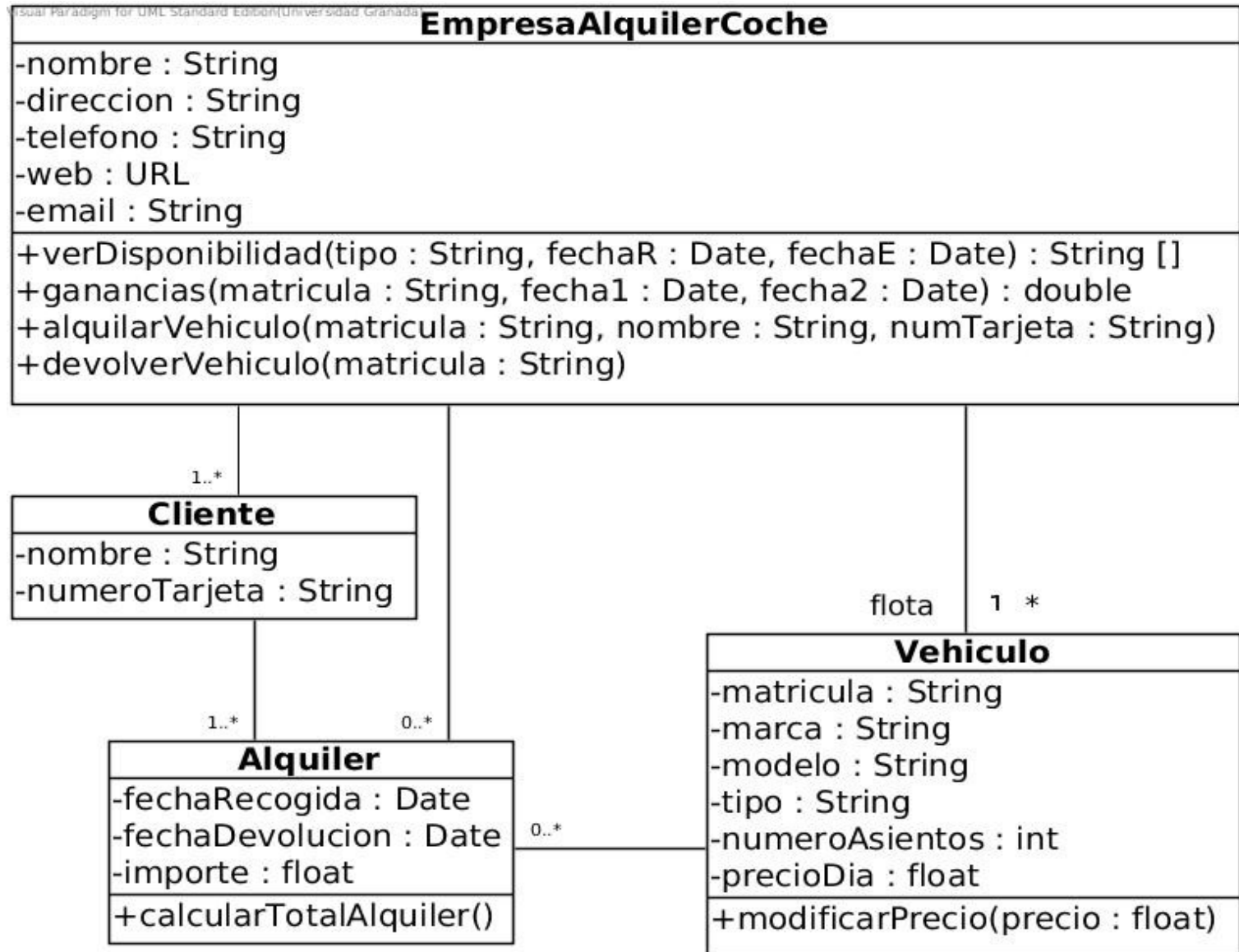


# 4. Modelando Diagramas de clases

## 4. Identificando Asociaciones

Una **empresa de alquiler de vehículos** posee una amplia **flota de vehículos**. La información relevante de cada vehículo es: matrícula, marca, modelo, el número de plazas y el tipo de vehículo. Cuando un **cliente** solicita un **vehículo** para su alquiler debe indicar la fecha de recogida, la fecha de devolución, el número de plazas necesarias, el tipo de vehículo y el número de tarjeta de crédito con que se abonará el alquiler. La empresa comprueba la disponibilidad en ese momento de algún **vehículo** del tipo solicitado, si lo hay, se le alquila al **cliente**, debiéndose registrar fecha de recogida, la fecha de devolución, el importe del alquiler, calculado a partir de precio por día del vehículo y los días que va a estar alquilado. En caso de no tener un vehículo disponible, se lo comunica al cliente y se anula la solicitud. La **empresa** desea conocer los **vehículos alquilados** en un determinado momento y un informe con las ganancias generadas por un determinado vehículo en un periodo de tiempo. Cuando un **cliente** devuelve el **vehículo** desaparece su información del sistema.

# 4. Modelando Diagramas de clases: Primera versión



## 4. Modelando Diagramas de clases: Refinamiento

Criterios de **refinamiento**:

- Requisitos funcionales existentes.
- Bajo acoplamiento entre clases.
- Buen nivel de cohesión en cada clase.

Tareas:

1. Incluir el nombre de las asociaciones y navegabilidades.
2. Refinar el diseño siguiendo los criterios anteriores.
3. Proponer un diseño alternativo.



**CONCLUSIÓN:** no existe un único diseño, puede haber más de uno que sea correcto, pero sí es importante tratar de conseguir el diseño que más se adecue a los criterios indicados anteriormente.