

# Diagrama de clases

Fundamentos de Ingeniería de Software / Análisis y Diseño de Software

Gastón Márquez  
Departamento de Informática  
Universidad Técnica Federico Santa María



# Motivación

- Fase de diseño
- Transición entre lo que el sistema *debe hacer* hacia *cómo* el sistema lo hará.
- Las clases se pueden derivar de los requerimientos.
- Sustantivos, potenciales clases
- Verbos, potenciales métodos
- UML para poder representar clases

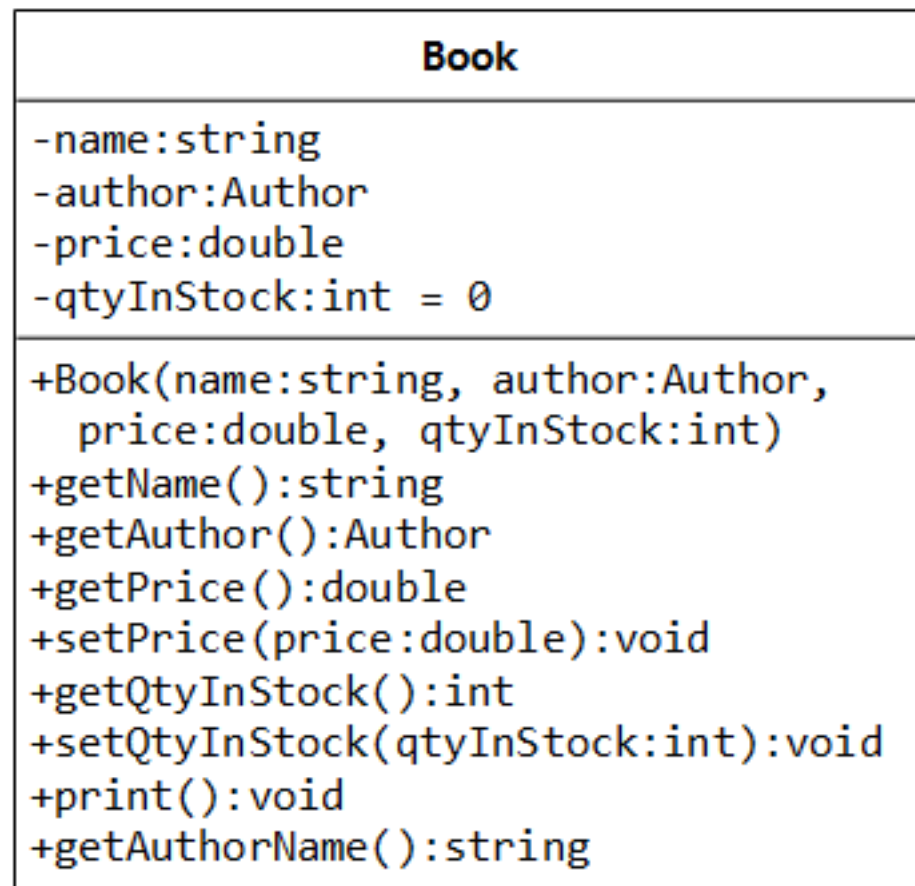
# Diagrama de clases [1]

- ¿Qué es un diagrama de clases UML?
  - Es una imagen de:
    - Las clases en un sistema OO
    - Métodos y atributos
    - Conexión entre clases
- ¿Qué *no* representa el diagrama de clases?
  - Detalles de cómo clases interactúan unas con otras
  - Detalles algorítmicos

# Diagrama de clases [2]

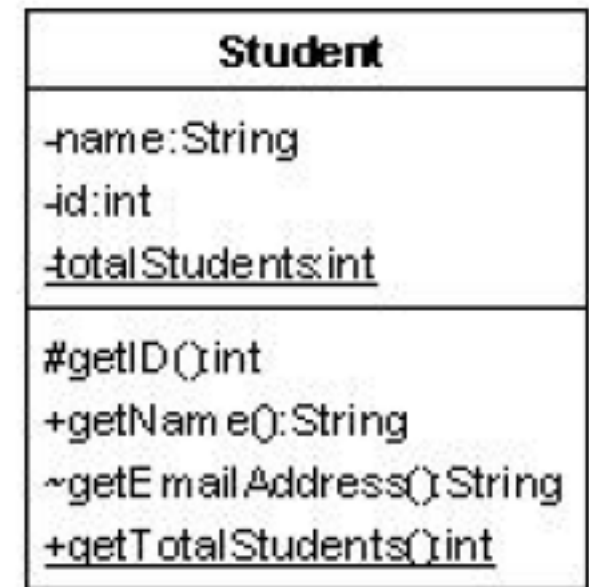
- El nombre de la clase se escribe al inicio
  - En caso de ser interface, se escribe `<<interface>>`
  - Se usa *cursiva*, cuando la clase es abstracta
- Los atributos (opcional)
  - Deben incluir todos los campos del objeto
- Los métodos (opcional)
  - Pueden omitir métodos triviales (get/set)
    - Pero no se omiten métodos de una interface
  - No debe incluir métodos de herencia

# Diagrama de clases [3]



# Diagrama de clases [4]

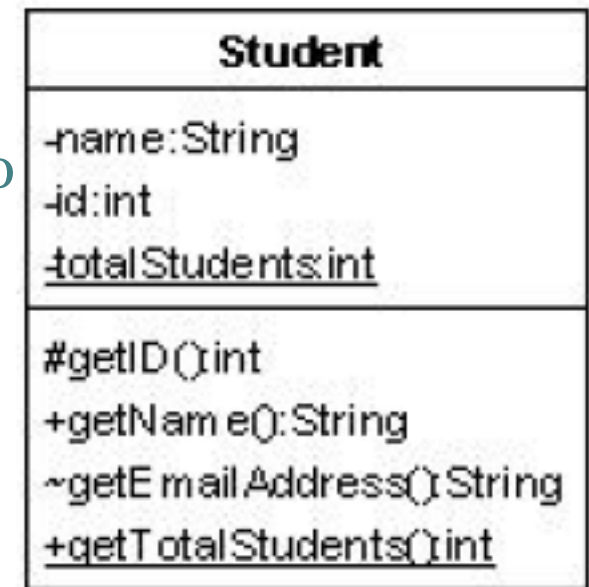
- Atributos
  - Visibilidad: tipo[count]=valor\_defecto
  - Visibilidad:
    - + public
    - # protected
    - - private
    - ~ package
    - / derived
  - Atributos subrayados: estáticos
  - Atributo derived: no están almacenados, pero pueden ser computados por valores de otros atributos
    - Por ejemplo:
      - -balance : double = 0.00



# Diagrama de clases [5]

- Métodos

- Visibilidad nombre (par.): tipo\_retorno
- Visibilidad:
  - + public
  - # protected
  - - private
  - ~ package
- Métodos subrayados: estáticos
- Parámetros → (nombre: tipo)
- Se omite el tipo de retorno en constructores y cuando el tipo de retorno es void
- Ejemplo de método
  - + distance(p1: Point, p2: Point): double





# Relaciones entre clases [1]

- Generalización: una relación de herencia
  - Herencia entre clases
  - Implementación de interface
- Asociación: relación de uso
  - Asociación
  - Agregación
  - Composición
  - Dependencia

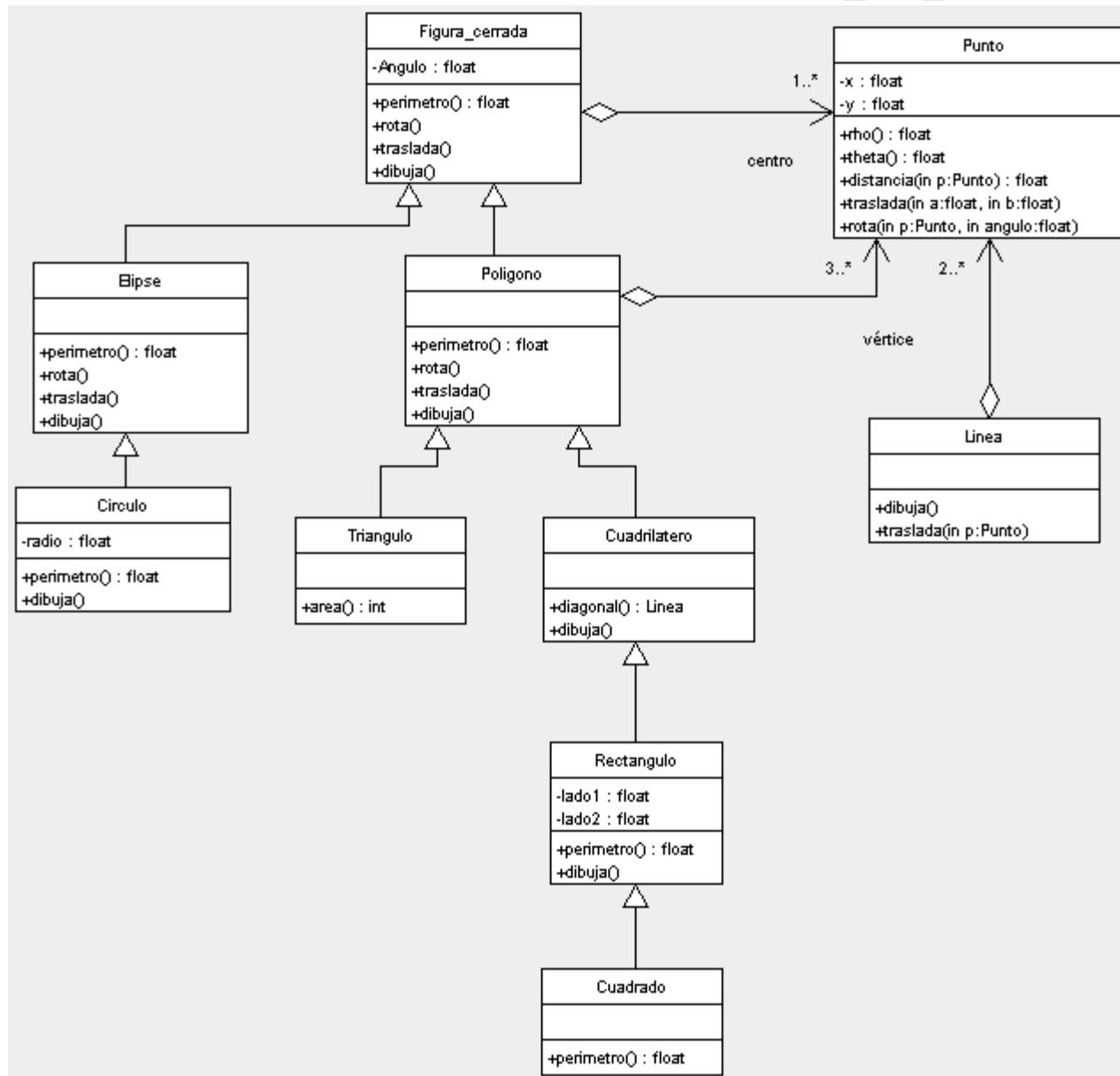




# Relaciones entre clases [2]

- Generalización (herencia)
  - La herencia relaciona dos objetos indicando que uno se deriva o es una extensión del otro, de tal forma que uno se puede entender como una generalización y el otro como una especialización de un concepto común.
  - El verbo asociado a esta relación es “*es un*”

# Relaciones entre clases [3]



# Relaciones entre clases [4]

- A partir de la figura anterior podemos observar:
  - Las clases ELIPSE y POLIGONO heredan atributos y métodos de la clase FIGURA CERRADA
  - Las clases CUADRILATERO y TRIANGULO heredan de la clase POLIGONO (y a través de esta clase heredan desde la clase FIGURA CERRADA).
  - La clase CUADRADO hereda de RECTANGULO y ésta de CUADRILATERO.



## Relaciones entre clases [4]

- A partir de la figura anterior podemos observar:
  - ¿Qué más observa usted?

## Relaciones entre clases [4]

- A partir de la figura anterior podemos observar:
  - La clase FIGURA CERRADA se compone de un punto que constituye el centro de la figura, en el plano XY
  - Asimismo POLIGONO se compone de un conjunto de vértices (al menos 3), que no son otra cosa que puntos en el plano XY



## Relaciones entre clases [5]

- Por otro lado, se dice que una clase es superclase de otra si la primera es padre de la segunda, siendo esta última una subclase de la primera.
- Por ejemplo en el diagrama, la clase CUADRADO es subclase de RECTANGULO, y ésta es superclase respecto de la clase CUADRADO, pero es una subclase respecto de CUADRILATERO.

# Relaciones entre clases [6]

```
public class RECTANGULO extends CUADRILATERO {
```

```
    // Atributos
```

```
    private float lado1;
```

```
    private float lado2;
```

```
    // Constructor
```

```
    public RECTANGULO(float a, float b) {
```

```
        lado1 = a;
```

```
        lado2 = b;
```

```
        ...
```

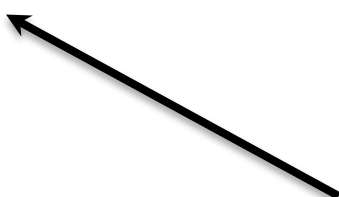
```
    }
```

```
    // Métodos
```

```
    public float perimetro() {...}
```

```
    public void dibuja() {...}
```

```
} /* fin clase RECTANGULO */
```



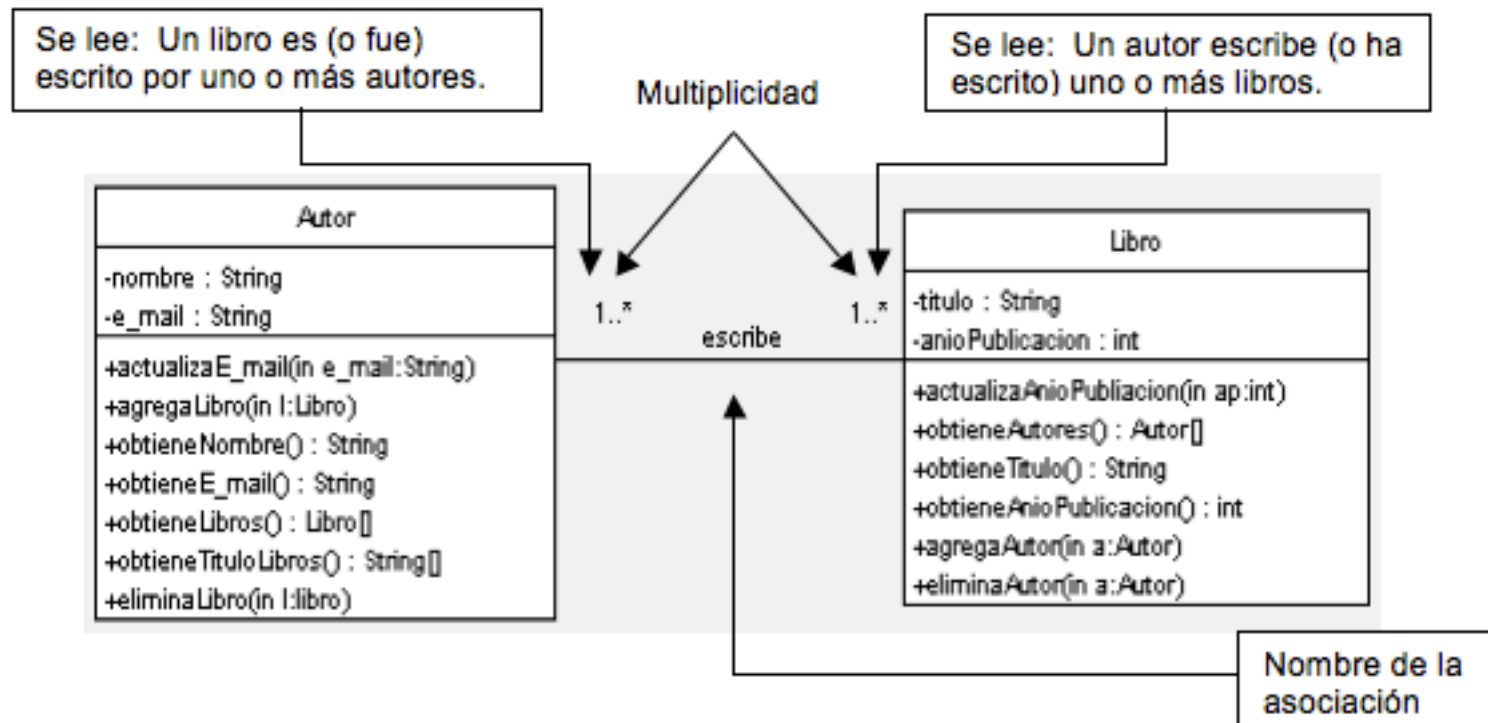
Indica que la clase RECTANGULO hereda atributos y métodos de la clase CUADRILATERO y, por su intermedio, de sus superclases.

# Relaciones entre clases [7]

- Asociación (relación de uso)
  - La *asociación* se produce entre dos o más objetos.
  - La más simple → asociación binaria.
  - Una asociación es la más general de las relaciones.



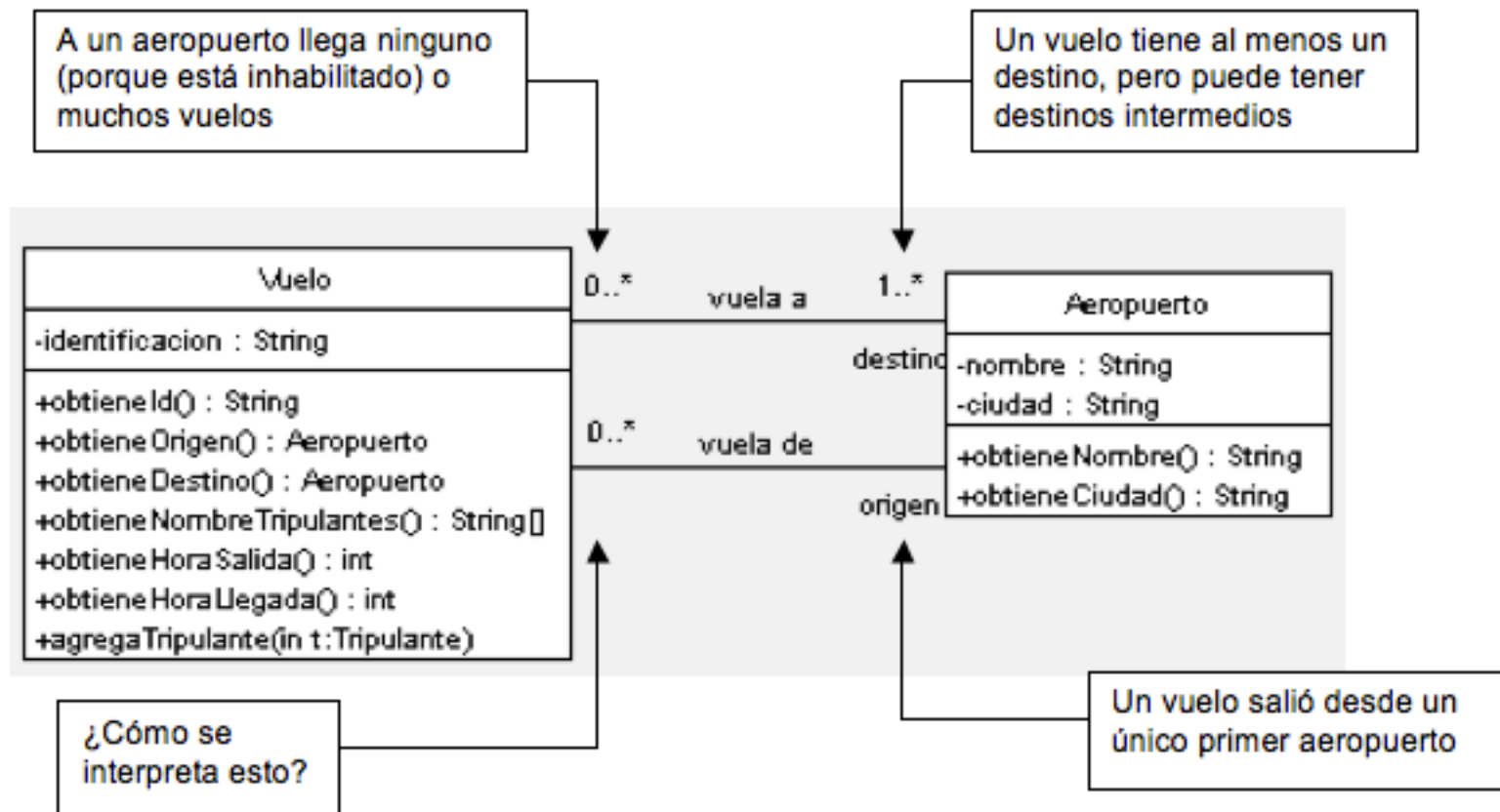
# Relaciones entre clases [7]



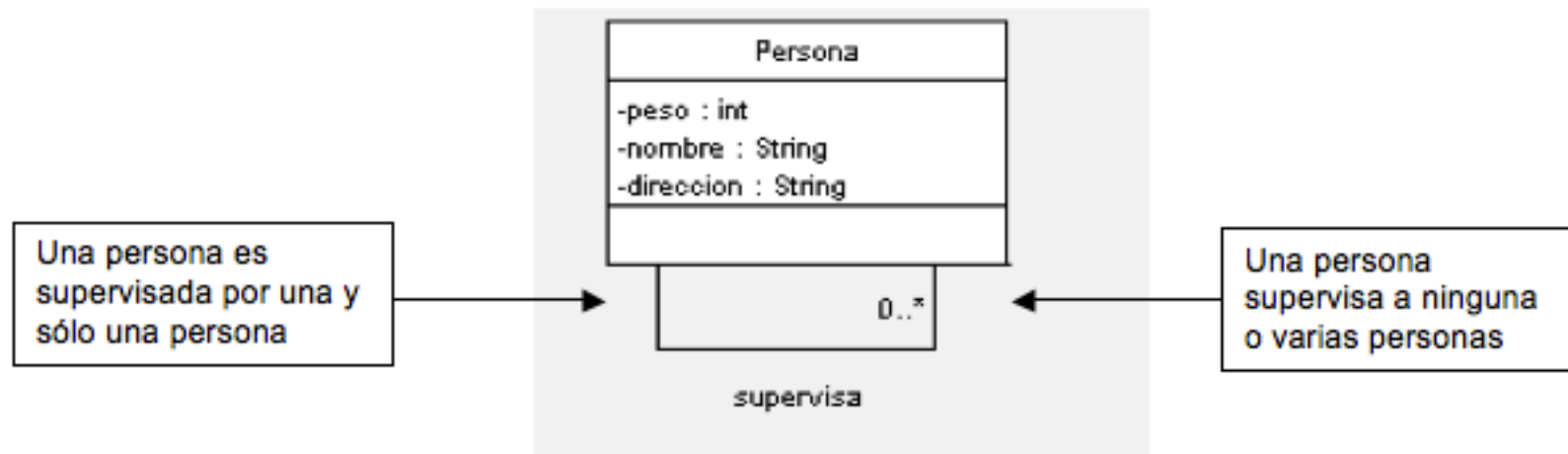
# Relaciones entre clases [7]

- Multiplicidad
  - uno y sólo uno (1..1)
  - 0 o uno (0..1)
  - uno o más (1..\*)
  - 0 o más (0..\*)

# Relaciones entre clases [7]



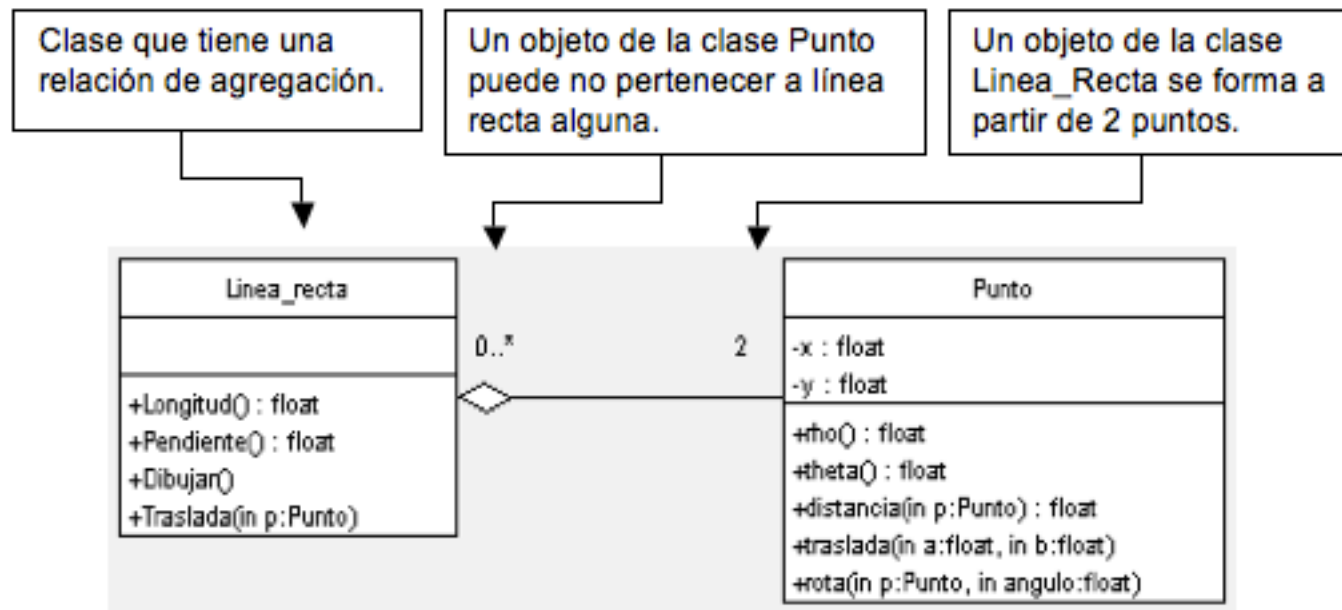
# Relaciones entre clases [7]



# Relaciones entre clases [8]

- Asociación (relación de uso)
  - Una *agregación* es una relación que permite indicar si un objeto se compone de otros objetos.
  - Por lo general, esta relación no lleva nombre en un diagrama.
  - Dichos objetos tienen un ciclo de vida independiente.

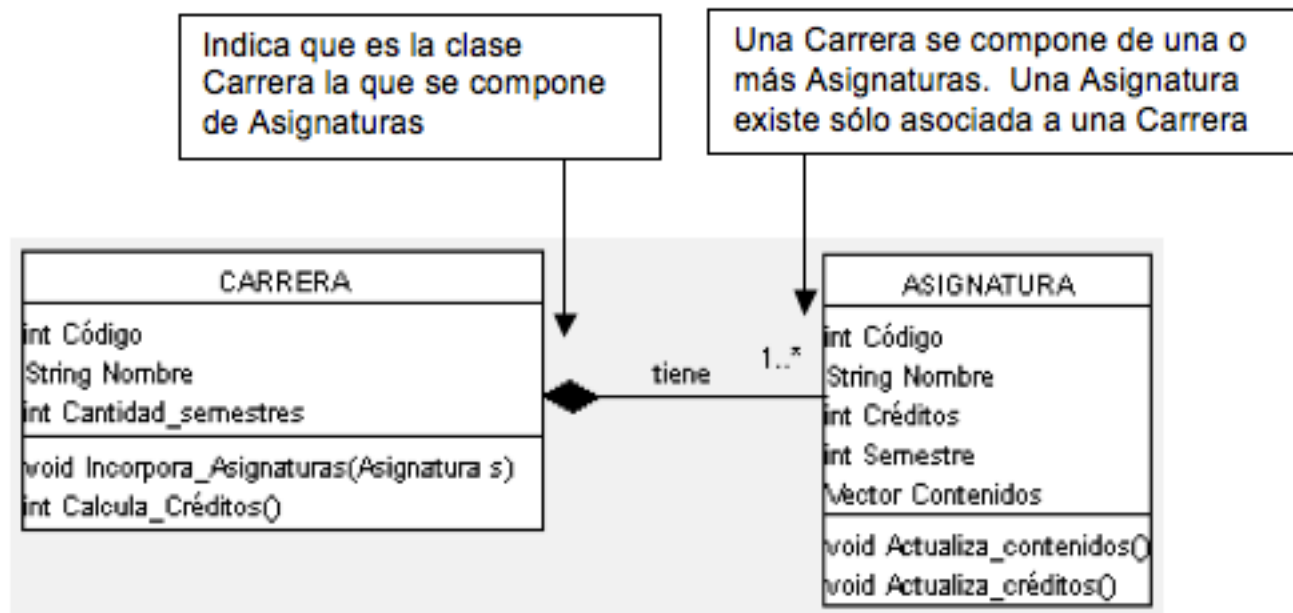
# Relaciones entre clases [8]



# Relaciones entre clases [9]

- Asociación (relación de uso)
  - Una *composición* es una relación semejante a la agregación, pero más estricta.
  - Por ejemplo, si el objeto A se compone del objeto B y B desaparece (por alguna razón), luego, A también.

# Relaciones entre clases [9]

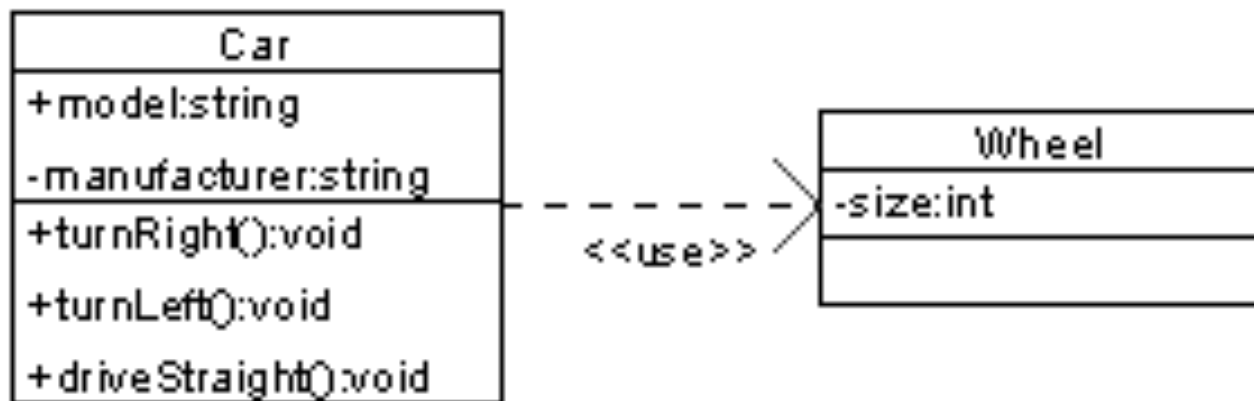




# Relaciones entre clases [10]

- Asociación (relación de uso)
  - La *dependencia* representa la usabilidad de la relación.
  - Si algo cambia en una clase A, necesariamente el cambio afectará a la otras que son dependientes de la clase A.

# Relaciones entre clases [10]



# Relaciones entre clases [11]

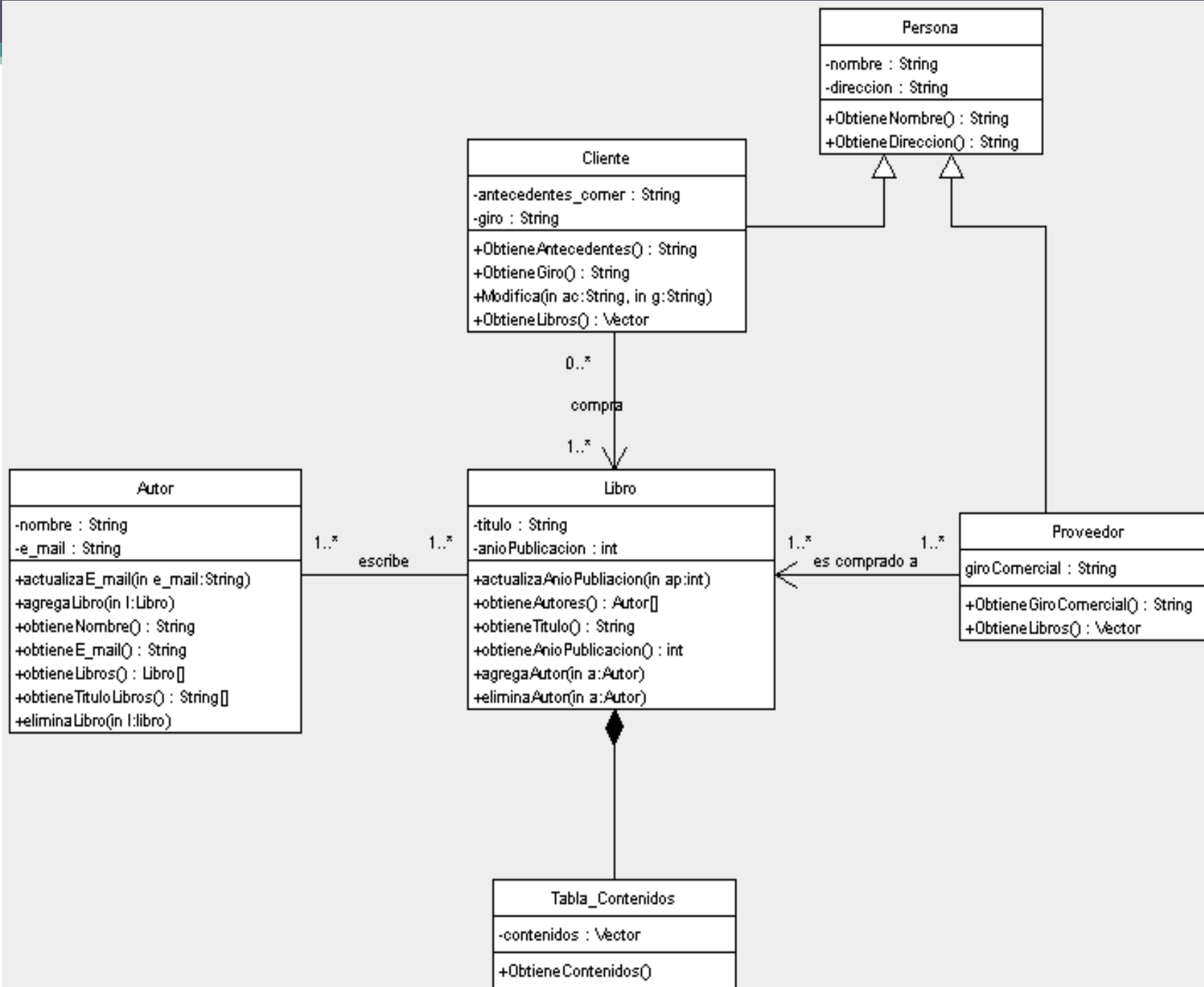
- Una interfaz es una colección de métodos abstractos y propiedades
- Es una especie de “menú” que indica qué se debe hacer pero no su implementación
- Las clases que utilicen interfaces describirán la lógica de los métodos
- Ordena el código
- Ciertas clases utilizan el mismo método
- Relaciones con clases que no necesariamente están relacionadas

# Relaciones entre clases [11]

```
interface Bicycle {  
    // wheel revolutions per minute  
    void changeCadence(int newValue);  
  
    void changeGear(int newValue);  
  
    void speedUp(int increment);  
  
    void applyBrakes(int decrement);  
}
```

# Relaciones entre clases [11]

```
class ACMEBicycle implements Bicycle {  
  
    int cadence = 0;  
    int speed = 0;  
    int gear = 1;  
  
    void changeCadence(int newValue) {  
        cadence = newValue;  
    }  
  
    void changeGear(int newValue) {  
        gear = newValue;  
    }  
  
    void speedUp(int increment) {  
        speed = speed + increment;  
    }  
  
    void applyBrakes(int decrement) {  
        speed = speed - decrement;  
    }  
  
    void printStates() {  
        System.out.println("cadence:" +  
            cadence + " speed:" +  
            speed + " gear:" + gear);  
    }  
}
```





# Referencias

- Apuntes de Diseño y Construcción de Algoritmos. A. Caro y M. Soto, UBB, 2007.
- Documentación oficial de UML, <http://www.uml.org>