UD7. Análisis y Diseño Orientado a Objetos

Índice

1. Introducción
2. Factores de calidad
   a) Acoplamiento
   b) Cohesión
3. UML
4. Diagrama de Clases
   a) Clases
   b) Asociaciones
      - Clase asociación
      - Composición vs agregación
      - Herencia
      - Realización
      - Dependencia
      - Estereotipos
5. Principios Generales de Diseño
   - Principio YAGNI
   - Principio DRY
   - Principio KISS (Keep It Simple Stupid)

1. Introducción

- En el **diseño** orientado a objetos un sistema se entiende como un **conjunto** de **objetos** que poseen **propiedades** y **comportamientos** y colaboran entre ellos para realizar tareas.
- **UML** (Unified Modeling Language) se ha convertido en un **estándard de facto**; se trata de un lenguaje de modelado a través de diagramas.

1.Introducción

1.1 Programación Orientada a Objetos (POO)

- El paradigma OO se basa en el **concepto** de **objeto**. Un objeto posee un **estado** y un **comportamiento.**
- Las **clases** son un concepto **estático** mientras que los **objetos** son entes **dinámicos** que existen en el tiempo y espacio durante la ejecución de un programa-

1.Introducción

1.2 Principios de la POO.

- **Abstracción;** Expresa las características esenciales de un objeto, donde se capturan sus comportamientos y que lo distinguen de los demás. El objetivo es obtener una descripción formal.

  Ej. coche

  características → *modelo, color, marca…*

  comportamientos → *accelerar, frenar, girar...*

1.Introducción

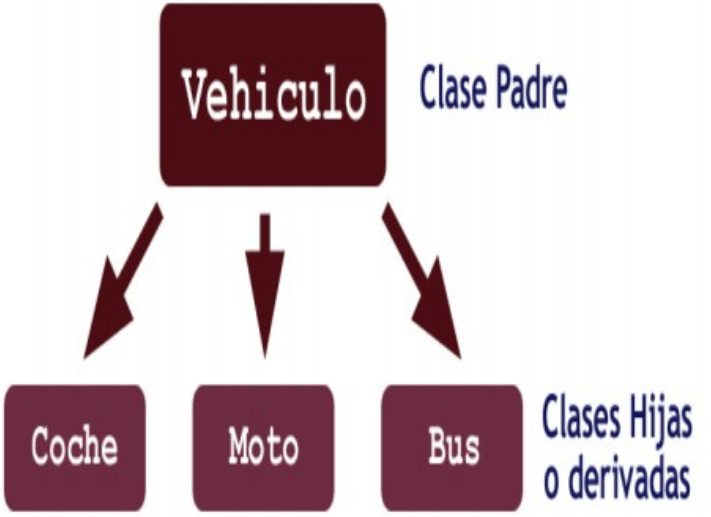
1.2 Principios de la POO.

- **Encapsulación;** Es el proceso de ocultar todos los detalles de un objeto que no contribuyen a sus características esenciales. Los objetos son **cajas negras** (sabemos que hace pero no como lo hace).

1.Introducción

1.2 Principios de la POO.

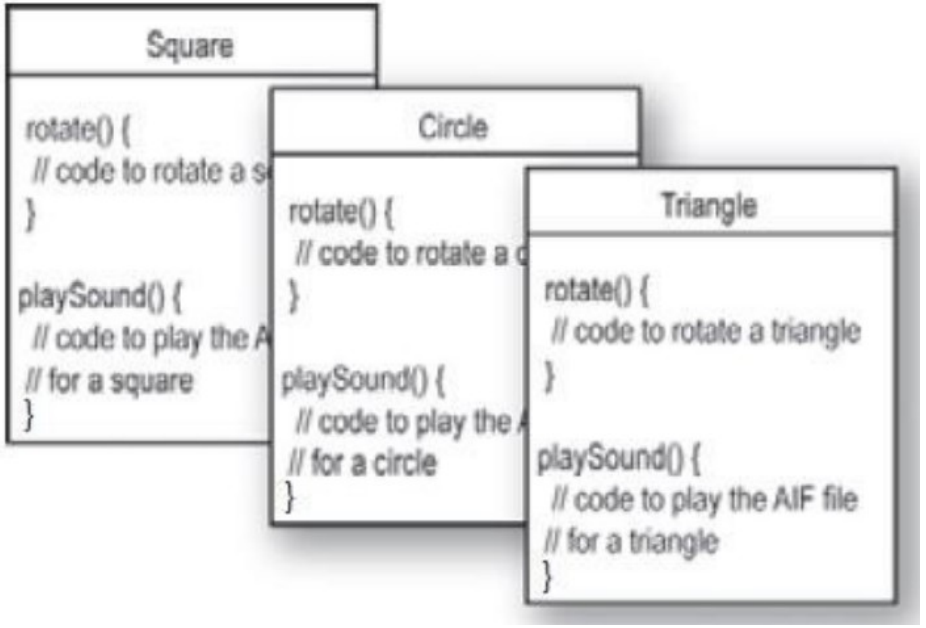
- **Modularidad;** El sistema se descompone en un conjunto de modulos o partes más pequeñas independientes.
- **Herencia;** Definición de clases a partir de otras de las cuales heredan todo el comportamiento y variables de la superclase.



1.Introducción

1.2 Principios de la POO.

- **Polimorfismo;** Consiste en un unir en un mismo nombre comportamientos diferentes.



2.Factores de calidad

2.1 Acoplamiento.

"Se define como el grado de interdependencia entre los módulos o unidades funcionales de un sistema".

- Es deseable tener **unidades de software** que sean **independientes** entre sí, **(bajo acoplamiento)** de esta forma ningún modulo tendrá que preocuparse de los detalles internos de los otros componentes.
- Podemos entender como unidades funcionales; una función, un método, una clase, una librería, una aplicación, un componente...

2. Factores de calidad

2.1 Acoplamiento.

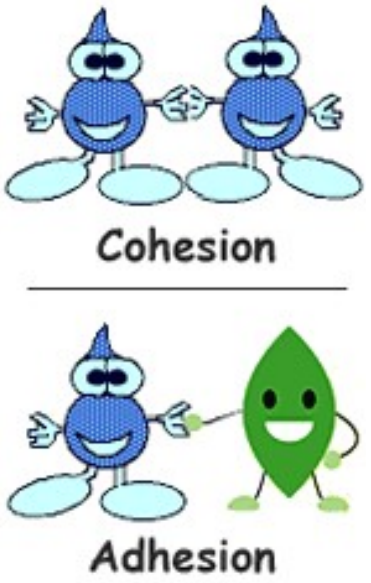
- El **acoplamiento** mide el **grado** en que una clase está **conectada** a otra, tiene **conocimiento** de otra, o de alguna manera **depende** de **otra**
- Lo ideal es que la **clase A** solo conozca de la **clase B** lo necesario para que pueda hacer uso de sus métodos.

2. Factores de calidad

2.1 Cohesión

"El nivel de cohesión de un módulo indica la conexión funcional entre sus elementos"

- Es **deseable** tener unidades de software con una **alta cohesión**.
  - Cohesión de método
  - Cohesión de clase



2. Factores de calidad

2.1 Cohesión

- El grado de cohesión indica si una clase tiene una función bien definida dentro del sistema. **(Single responsibility principle)**
- Una prueba fácil de cohesión consiste en examinar una clase y decidir si **todo** su **contenido** está **directamente relacionado** con el nombre de la clase y descripción de la misma.

2. Factores de calidad

2.1 Cohesión

- ¿Que pasaría si?
  1. Creamos clase A que necesita un acceso a BD y lo realiza dentro de la clase.
  2. Creamos clase B que también necesita acceso a la BD.
  3. Decidimos cambiar el SGBD o el usuario y password.

2. Factores de calidad

- Una alta cohesión y un bajo acoplamiento:
  - **Reducirá** el **ruido** en el sistema; propagación de errores.
  - **Facilitará** el **mantenimiento** y la **modificación**.
  - **Facilitará** la legibilidad y el entendimiento**.**

3. UML

- **Lenguaje gráfico** para **visualizar**, **especificar** y **documentar** las diferentes **etapas** que comprenden el **desarrollo** de **software**.
- Es utilizado para **modelar** tanto sistemas **software** como **hardware**.
- Existen **2** grandes **versiones**:
  - **UML 1.X**; *publicada a finales de los 90*
  - **UML 2.x**; publicada a finales del 2005
  - **UML 2.5;** publicada en 2015 (Actual)

3. UML

- Define 13 tipos de diagramas clasificados en:
  - **Diagramas de estructura**: Se centran en los elementos que deben existir en el sistema modelado.
  - **Diagramas de comportamiento:** Utilizado en el análisis de requisitos. Se centran en lo que debe suceder en el sistema.
  - **Diagramas de interacción:** Se centran en el flujo de control y de datos entre los elementos del sistema.

3. UML

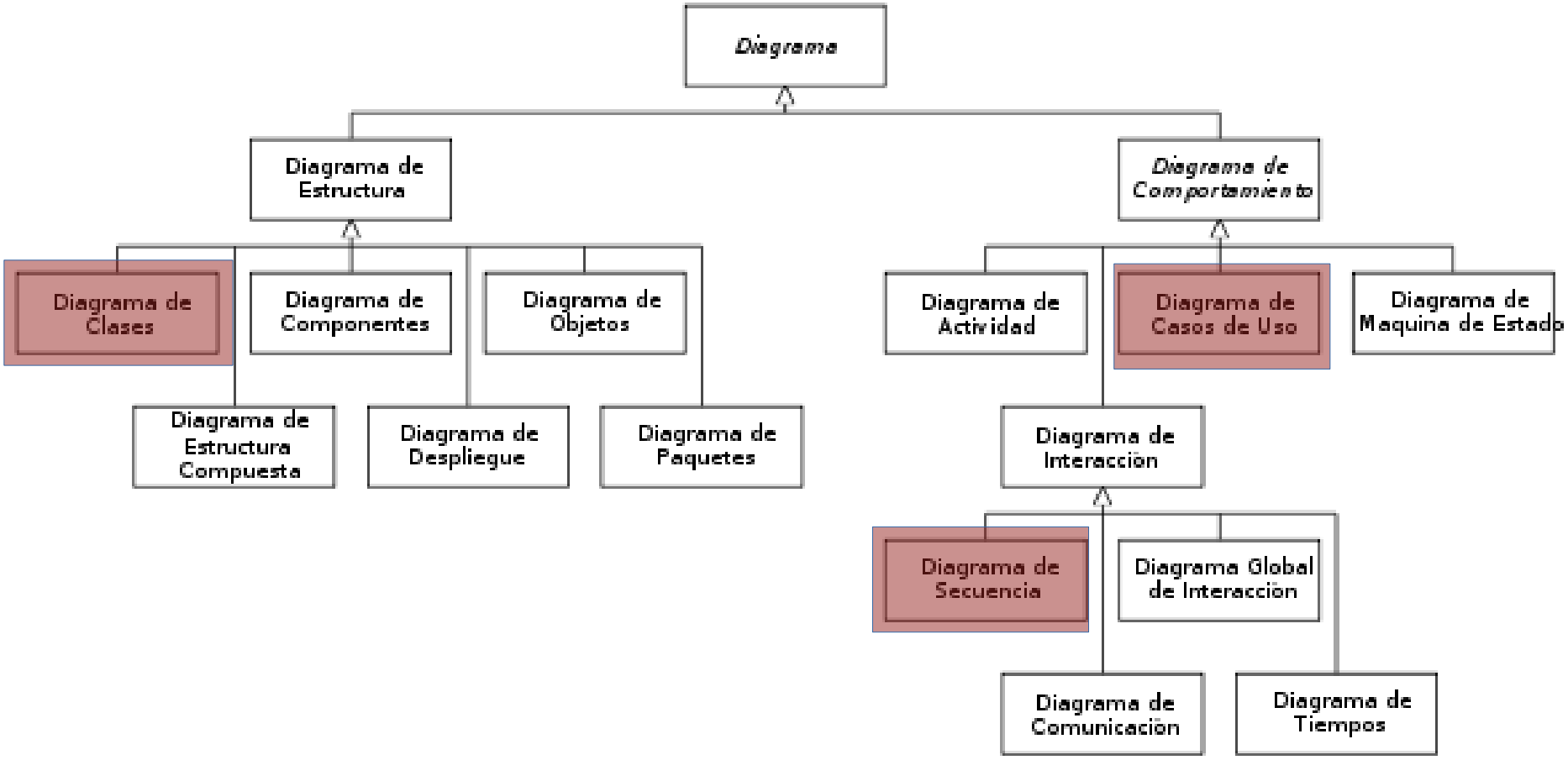


Diagrama
Diagrama de Estructura
Diagrama de Comportamiento
Diagrama de Clases
Diagrama de Componentes
Diagrama de Objetos
Diagrama de Estructura Compuesta
Diagrama de Despliegue
Diagrama de Paquetes
Diagrama de Actividad
Diagrama de Casos de Uso
Diagrama de Maquina de Estado
Diagrama de Interaccion
Diagrama de Secuencia
Diagrama Global de Interaccion
Diagrama de Comunicacion
Diagrama de Tiempos

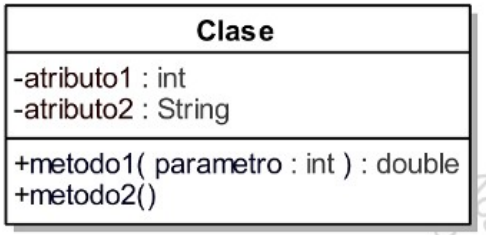
4. Diagrama de Clases

- **Muestra** las diferentes **clases** que **forman** el **sistema** y como se relacionan unas con otras.
- Se utiliza para modelar la **vista estática** de un diseño.
- Se pueden "*construir sistemas ejecutables"* mediante herramientas **CASE.**
- Un diagrama de clases esta compuesto por:
  - **Clases**: atributos, métodos y visibilidad.
  - **Relaciones**; asociación, herencia, agregación, composición, dependencia.

4. Diagrama de Clases

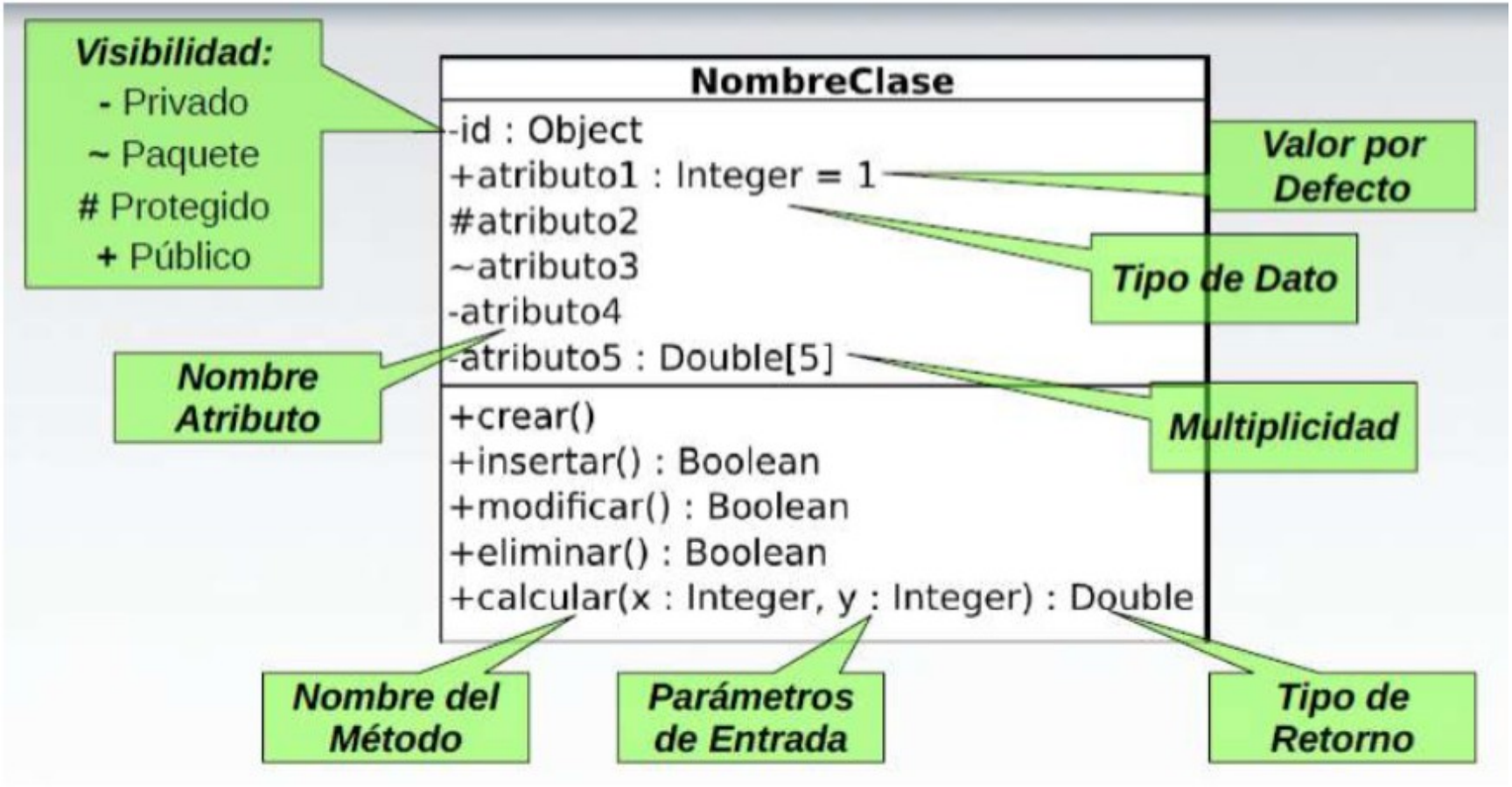
4.1 Clases

- La parte superior contiene el nombre de la clase, la parte intermedia los atributos y la inferior los métodos u operaciones.
- Tanto los métodos como los atributos pueden ser:
  - **Private**: Accesible por la propia clase (-)
  - **Protected**: Accesibles por las subclases (#)
  - **Public**: Accesibles tanto desde dentro como clase. (+)
  - **Package:** Accesibles desde el mismo paquete (~)



4. Diagrama de Clases

4.1 Clases



4. Diagrama de Clases

4.2 Asociaciones

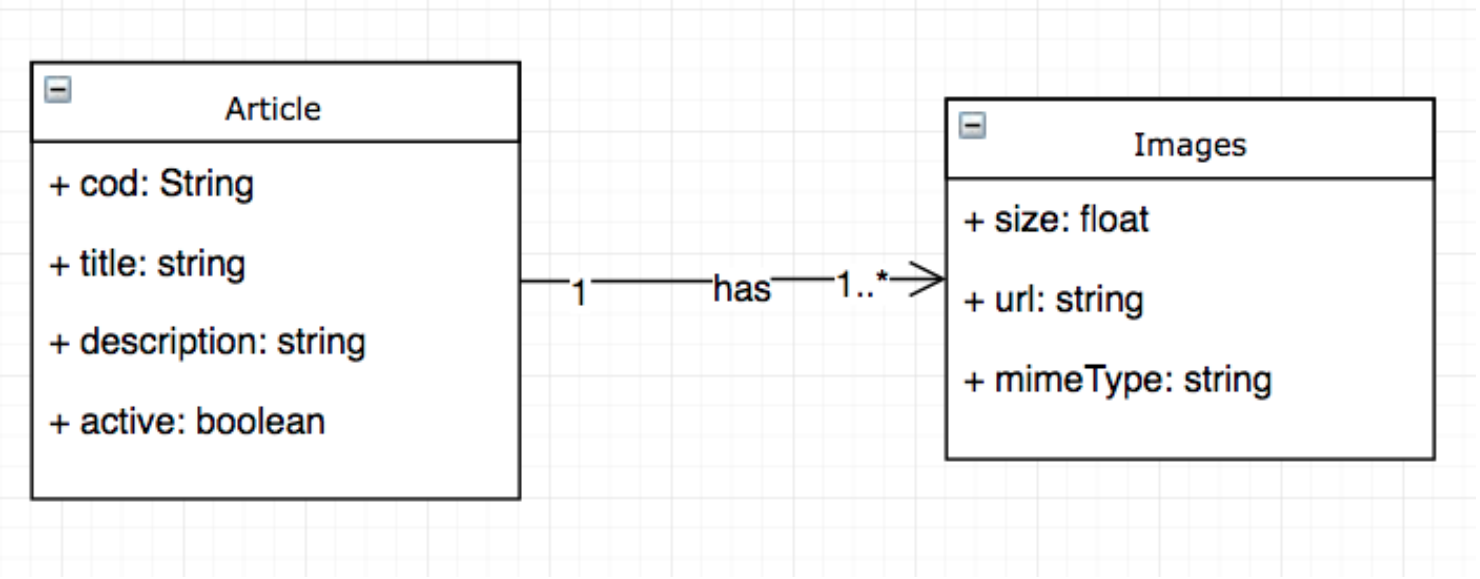
- En el mundo real los **objetos** están vinculados o **relacionados entre sí**. Estos vínculos se corresponden con asociaciones entre los objetos.
- Estas asociaciones tienen definida una **cardinalidad/multiplicidad.**

Notación	Cardinalidad / multiplicidad
0..1	Cero o una vez
1	Una y solo una
*	Cero o varias veces
1..*	De una a varias veces
M..N	Entre M y N veces
N	N veces

4. Diagrama de Clases

4.2 Asociaciones

- Cada **clase juega** un papel o **rol** que se **indicará** en la parte **superior** de la **línea** que une las dos clases.

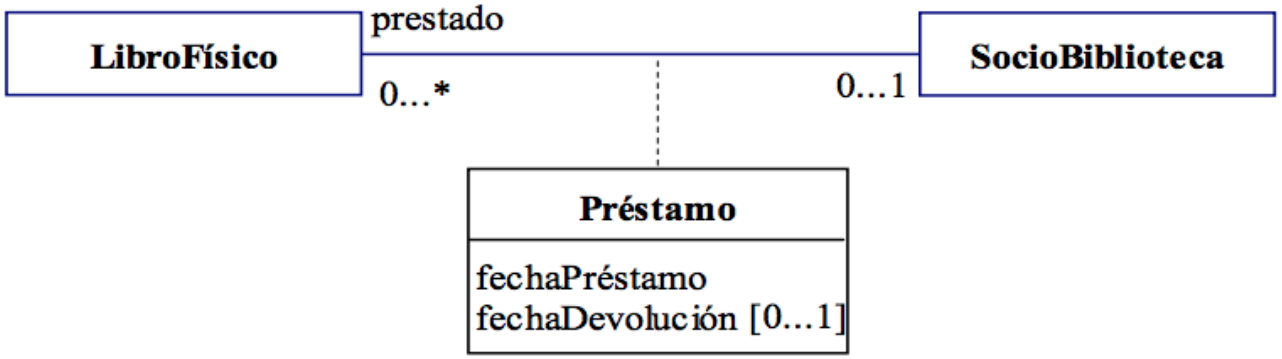


- Las asociaciones pueden ser **unidireccionales**, **bidireccionales o reflexivas**

4. Diagrama de Clases

4.2.1 Clase asociación

- Una **asociación** entre **2 o más clases** puede **llevar información** necesaria para esa asociación.
- Se crea una nueva **clase** que va **ligada a** la **relación**.
- Solo puede **existir** mientras la **relación** se **mantenga**.

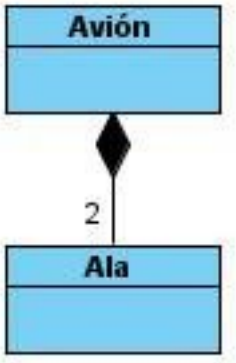


4. Diagrama de Clases

4.2.2 Composición vs agregación

- Se trata de relaciones en las cuales en las cuales una de las **clases** representa un **todo** y la **otra** representa **parte** de ese todo.

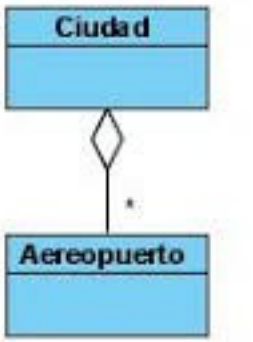
  - **Composición:** los componentes constituyen una parte del objeto compuesto y por tanto no pueden existir sin el objeto compuesto ni ser compartidos.



4. Diagrama de Clases

4.2.4 Composición vs agregación

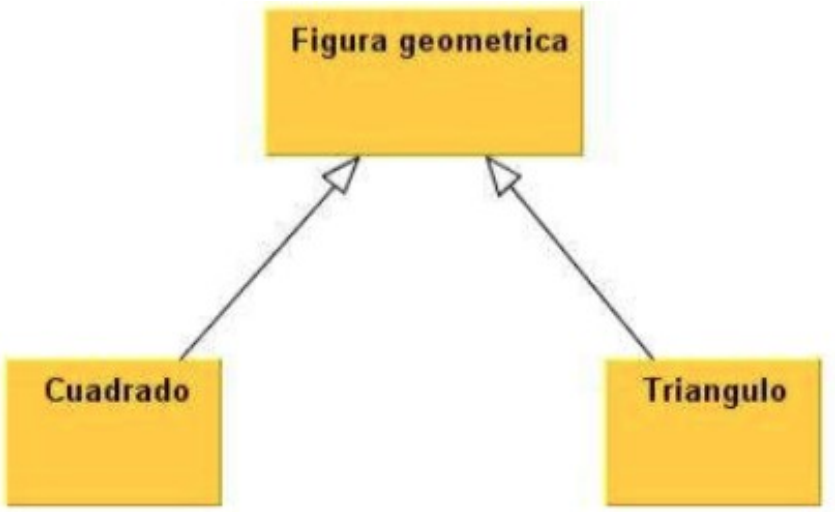
- **Agregación:** Se trata de una **composición débil**. Los componentes pueden ser compartidos por varios compuestos y la destrucción del compuesto no implica la destrucción de sus componentes.



4. Diagrama de Clases

4.2.3 Herencia

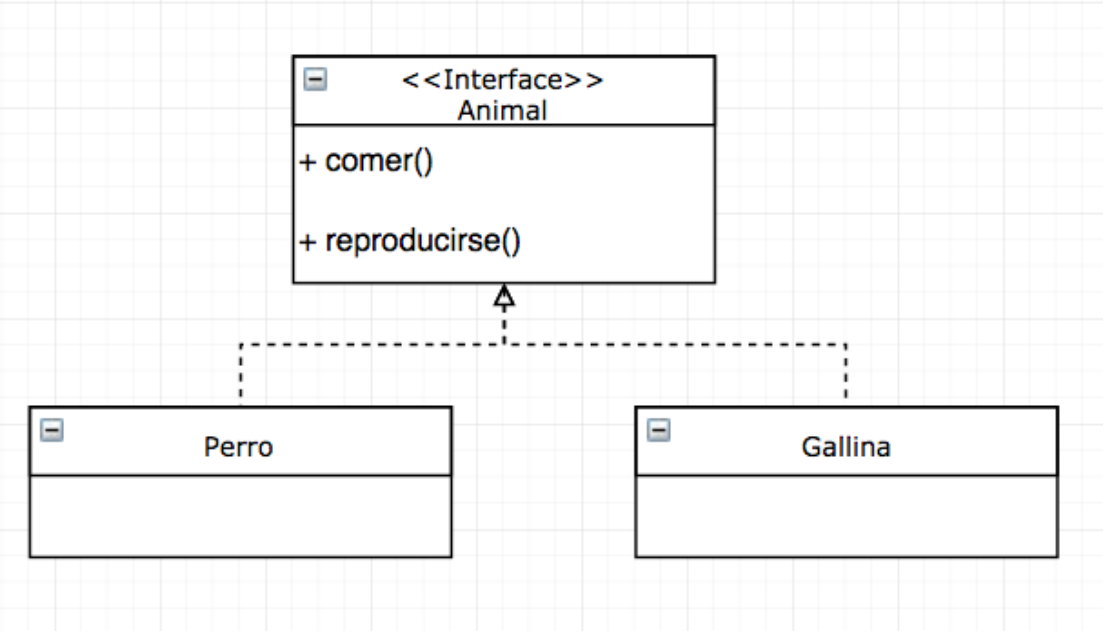
- Indica que una subclase **hereda** los **métodos** y **atributos** especificados por una Súper Clase.
- La Subclase poseerá las características y **atributos visibles** de la **Súper Clase** (**public y protected**).



4. Diagrama de Clases

4.2.5 Realización

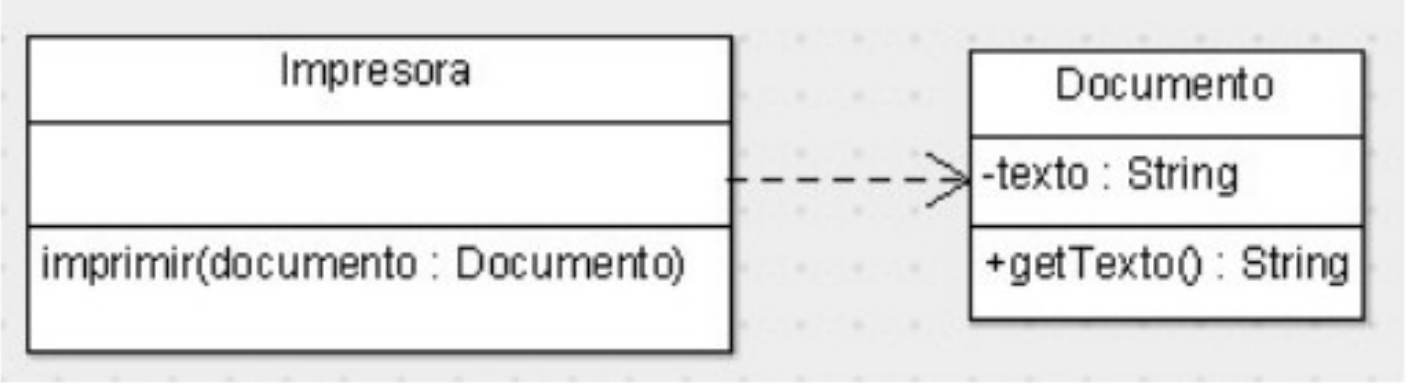
- Se trata de la **relación de herencia** existente entre una clase interfaz y la subclase que la implementa



4. Diagrama de Clases

4.2.6 Dependencia

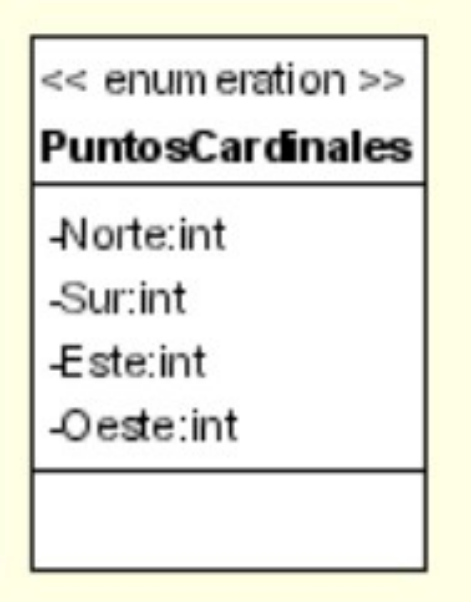
- Caso específico de asociación que se establece entre 2 clases cuando **una clase usa a la otra**.



4. Diagrama de Clases

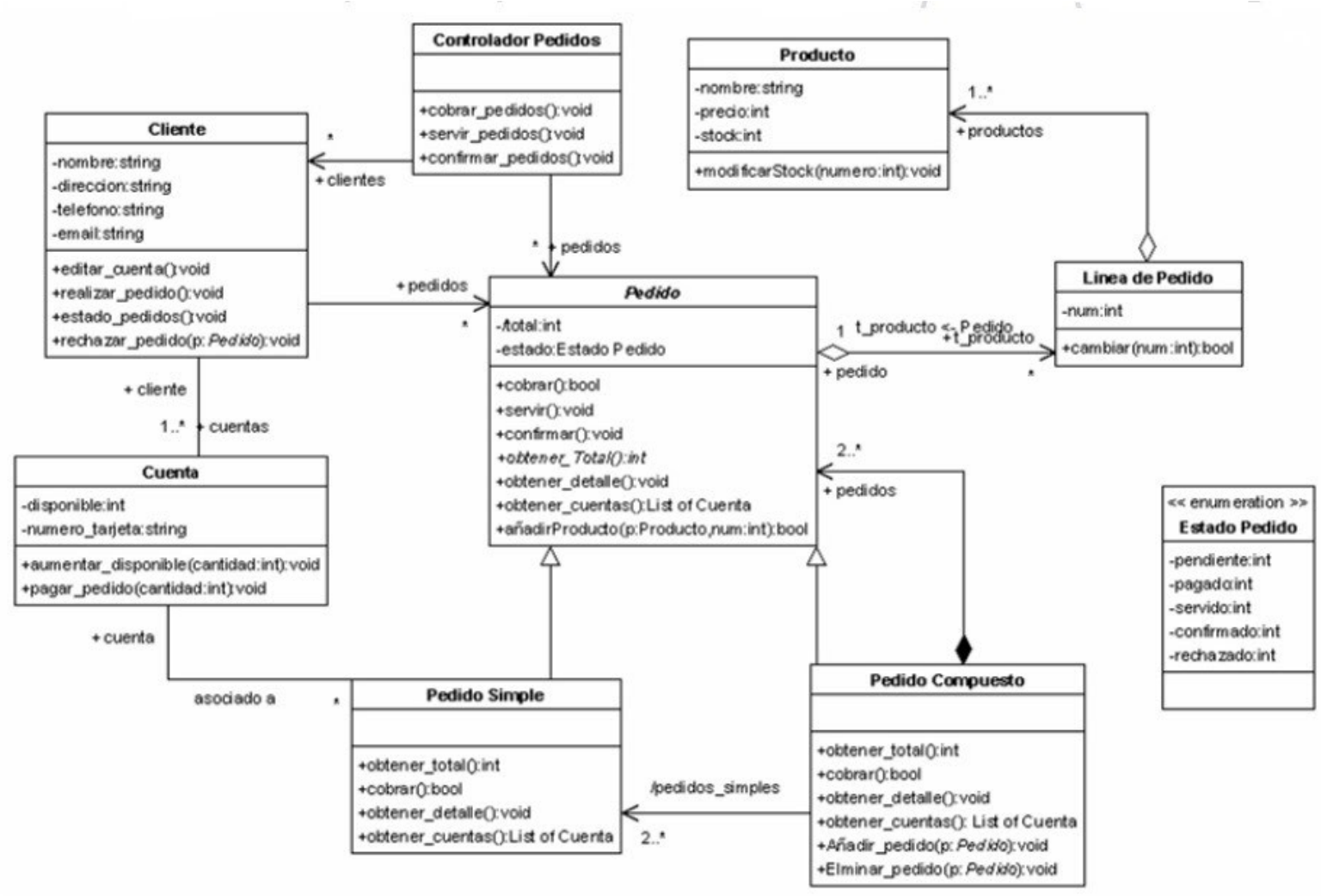
4.3 Estereotipos

- Permite definir **nuevos elementos** de modelado. UML, creando un metamodelo.
- Generalmente el nombre se pone entre comillas francesas **<<>>**



4. Diagrama de Clases

4.4 Ejemplo



5. Principios Generales de POO

5.1 Principio YAGNI *(You aren´t gonna need it)*

"No debemos agregar funcionalidad hasta que se considere estrictamente necesario"

- Las características innecesarias son un inconveniente debido a:
  - El tiempo gastado que se toma para la adición, la prueba o la mejora de funcionalidad innecesaria.
  - Conduce a la hinchazón de código y el software se hace más grande y más complicado.

5. Principios Generales del POO

5.2 Principio DRY *(Don´t Repeat yourself)*

" Cada pieza de conocimiento debe tener una única, inequívoca y autoritativa representación en un sistema"

- La modificación de un **elemento individual** de un sistema no debe requerir cambios en diferentes elementos

5. Principios Generales del POO

5.3 Principio KISS *(Keep it simple stupid)*

"La simplicidad debe ser un objetivo clave del diseño, y cualquier complejidad innecesaria debe evitarse"

- **Debemos evitar**:
  - Complejos algoritmos generalistas para situaciones muy concretas.
  - Complejos algoritmos muy eficientes cuando no hay necesidad.
  - Nombres de métodos con extraños nombres abstractos.
  - Enormes jerarquías de clases que no hacen nada
  - ....