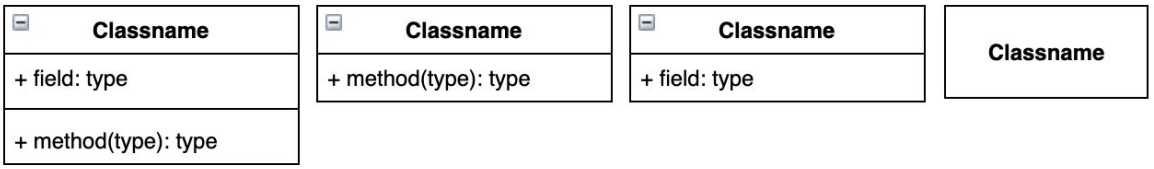
Apuntes diagramas de clases

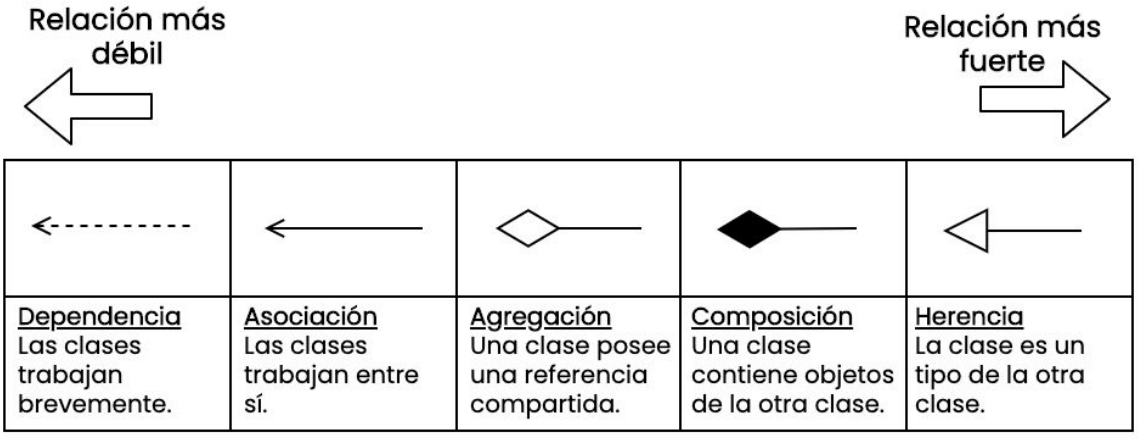
Adaptación del contenido hecho por Jose Ignacio Benedetto para el curso IIC2113 2019-2.

Los diagramas de clase buscan describir los distintos objetos presentes en un sistema, cómo interactúan y el conjunto de atributos y métodos de cada uno de ellos.

Rectángulos para representar las clases:



Líneas para representar las relaciones:



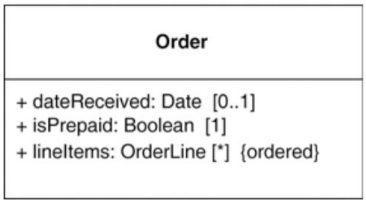
Texto para multiplicidad:

Multiplicidad	Opción	Cardinalidad
0..0	0	La colección debe estar vacia
0..1		Una o cero instancias
1..1	1	Exactamente una instancia
0..*		Cero o más instancias
1..*		Al menos una instancia
5..5	5	Exactamente 5 instancias
m..n		Al menos m, pero no más de n

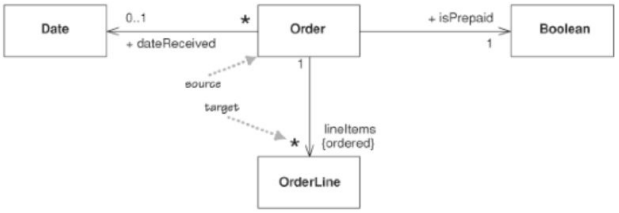
Propiedades (Property)

Análogos a campos o atributos en lenguajes de programación. Un property puede ilustrarse como atributo o asociación

- Atributo: declarado directamente dentro del cuadrado de la clase
- Asociación: declarado como asociación etiquetada entre dos clases



Properties como atributos



Properties como asociaciones

Operaciones (Operation)

Análogos a métodos de clase en lenguajes de programación. Curiosamente, no existe forma de representar funciones top-level en UML.

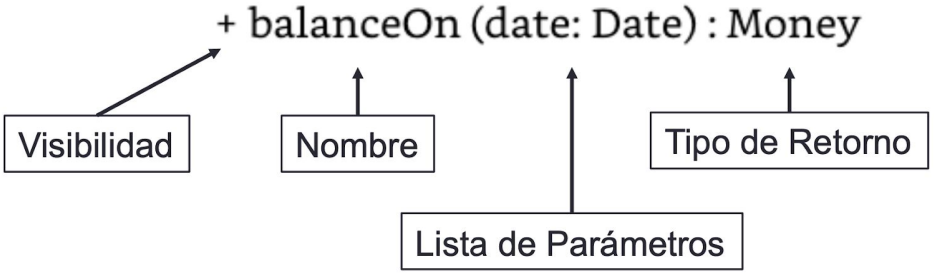
Notación estándar:

```
[Visibilidad][Nombre] ([Lista-de-Parámetros]):
      [Tipo-de-Retorno] {[Property-Modifier]}
```

Parámetro:

```
[Dirección][Nombre]:[Tipo] = [Default]
```

Ejemplo:



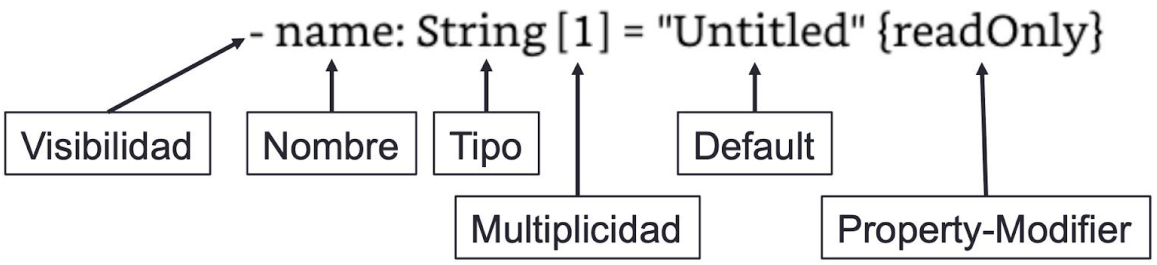
Atributos

Solo el nombre es obligatorio. Todos los demás elementos de la notación son opcionales.

Notación estándar:

```
[Visibilidad][/][Nombre]:[Tipo][`[’Multiplicidad`]’] =
      [Default] {[Property-Modifier]}
```

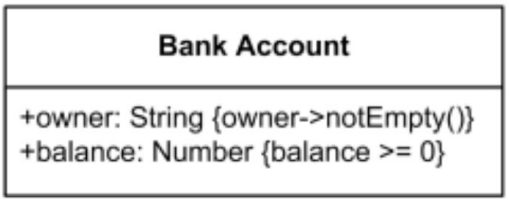
Ejemplo:



Property-Modifiers:

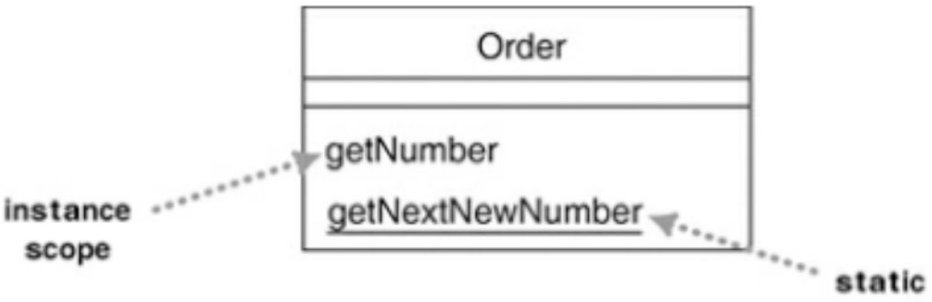
Modifier	Description
id	Property is part of the identifier for the class which owns the property.
readOnly	Property is read only (isReadOnly = true).
ordered	Property is ordered (isOrdered = true).
unique	Multi-valued property has no duplicate values (isUnique = true).
nonunique	Multi-valued property may have duplicate values (isUnique = false).
sequence (or seq)	Property is an ordered bag (isUnique = false and isOrdered = true).
union	Property is a derived union of its subsets.
redefines *property-name*	Property redefines an inherited property named ***property-name***.
subsets *property-name*	Property is a subset of the property named ***property-name***.
property-constraint	A constraint that applies to the property

Ejemplo de property-constraints:

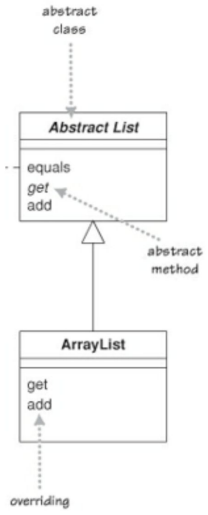


Modificadores adicionales

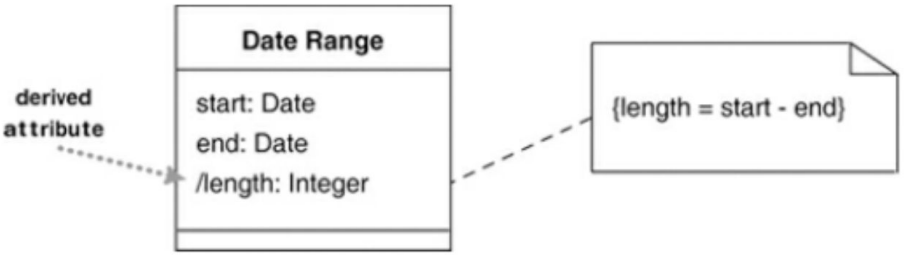
- Atributos o métodos estáticos: <u>subrayar</u>.



- Atributos, métodos o clases abstractas: *itálica*.



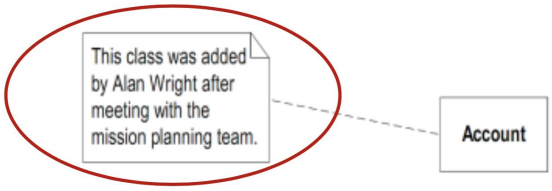
- Atributos derivados (se calculan en runtime): anteponer '/'



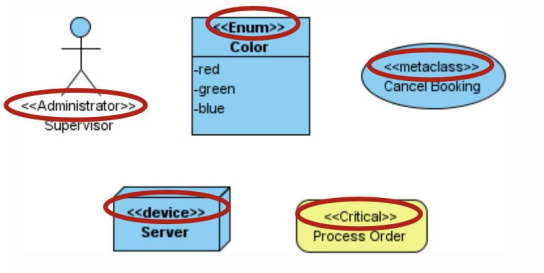
Comentarios y Estereotipos / Keywords

- Comentarios: Permiten especificar información adicional de un objeto UML en formato de texto libre.
- Estereotipos y Keywords:

- Análogo a una etiqueta, permite aportar más información semántica o de implementación acerca de un componente UML.
- La única diferencia entre ambos es que Keywords se usan para componentes del estándar, y estereotipos se utilizan en extensiones de UML definidas en perfiles.

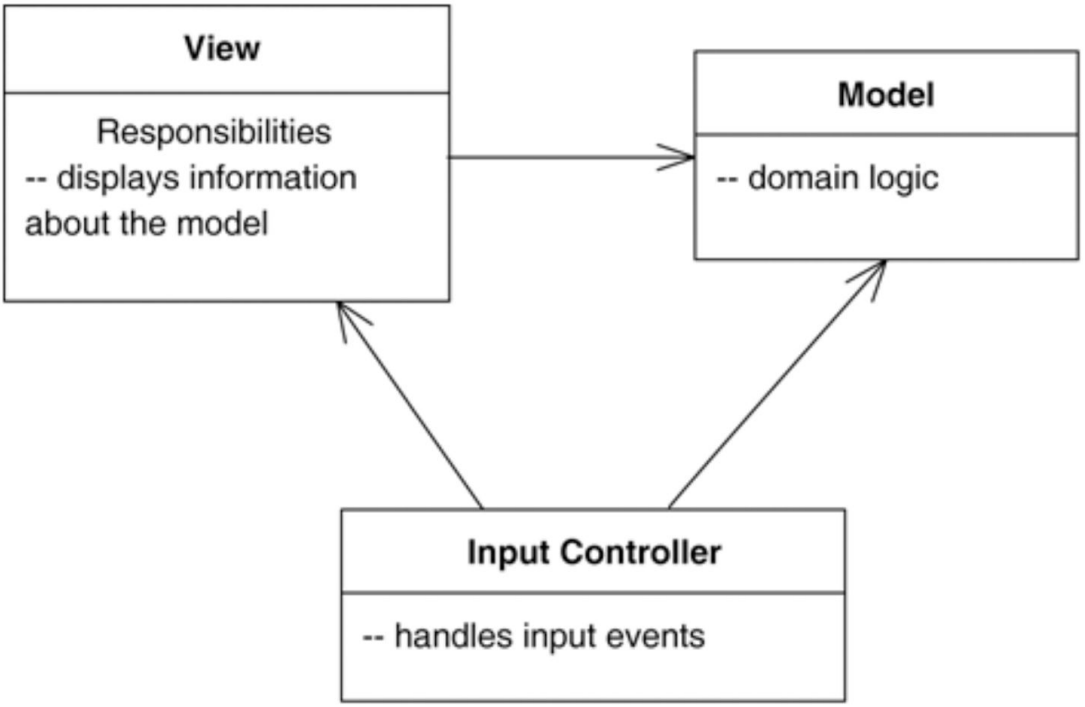


Ejemplo de comentario



Ejemplo de estereotipos y keywords

Ejemplo de comentarios dentro de contenedores:



Algunas keywords:

auxiliary	Class	A class that supports another class, typically by implementing additional logic. The other class may have the «focus» stereotype.
call	Dependency	The client class calls the operations of the supplier.
create	Dependency	The client class creates instances of the supplier.
create	Message	The sender creates the receiver.
create	Operation	This operation is a constructor.
derive	Dependency	The client element is computed completely or partly from the supplier.
destroy	Operation	The operation destroys its instance.
document	Artifact	A «file» that is not a source or an executable.
entity	Component	The component represents a business concept.
executable	Artifact	An executable «file».
file	Artifact	A physical file.
focus	Class	A class defining the core business logic, that is supported by several «auxiliary» classes.
framework	Package	This package defines a reusable design pattern.
implement	Component	The implementation of a «specification».
implementationClass	Class	The class describes an implementation, and each runtime instance has one fixed implementation class. Contrast with «type».
instantiate	Dependency	The client creates instances of the supplier.
library	Artifact	A library «file».
metaclass	Class	Instances of this class are also classes.
modelLibrary	Package	Contains model elements intended to be reused by importing packages. Typically defined as part of a profile, and imported automatically by application of the profile.

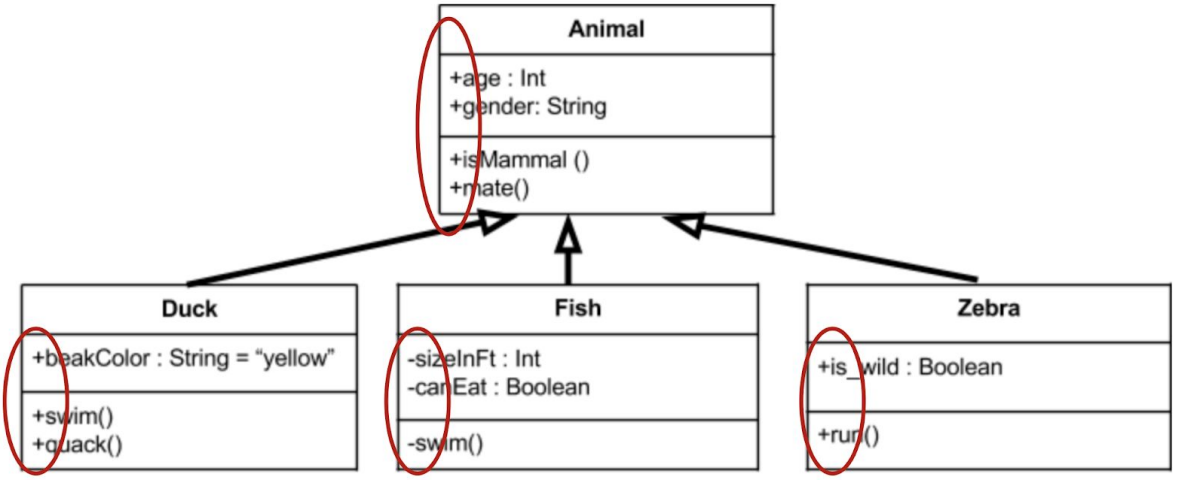
process	Component	A transaction-based component, or one that carries a thread.
realization	Class, Interface, Component	Describes an implementation.
refine	Dependency	The client class, component, or package provides more information about the specification or design than the supplier.
responsibility	Dependency	The Comment at the supplier end of the Dependency defines the responsibilities of the client class or component.
script	Artifact	An interpretable «file».
send	Dependency	The source Operation sends the target Signal.
service	Component	A stateless component.
source	Artifact	A compilable «file».
specification	Class, Interface, Component	Defines the behavior of a component or object without defining how it works internally.
subsystem	Component	A part of a large system. A subsystem on a use case diagram is a component with the subsystem stereotype.
trace	Dependency	The client element is part of the design that realizes the supplier. The two ends of this dependency are typically in different models. One of these models is a realization of the other.
type	Class	Specifies the behavior of an object without stating how it is implemented. An object is a member of a type if it conforms to the specification.
utility	Class	A collection of static functions. The class has no instances.

Visibilidad

UML ofrece la opción de etiquetar cada atributo u operación según su visibilidad hacia otras clases. Hay cuatro tipos de visibilidad:

- Public (+): accesible por todos
- Protected (#): accesible solo por descendientes
- Private (-): accesible solo por si mismo
- Package (~): accesible solo por clases del mismo módulo o paquete

* Notar que la semántica exacta de estas etiquetas puede variar sutilmente según el lenguaje de programación.

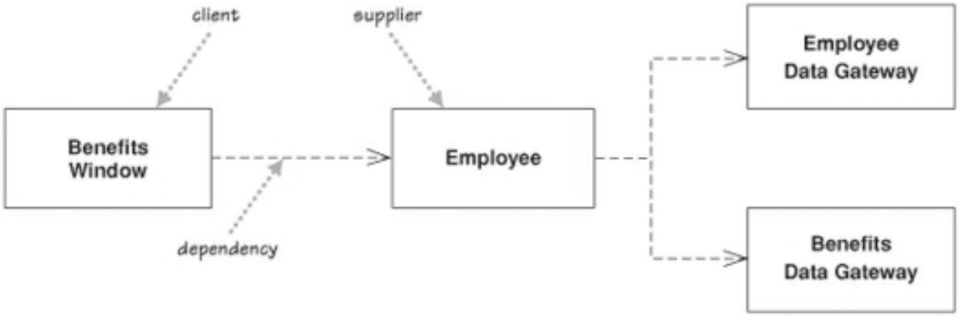


Dependencias

Se dice que existe una relación de dependencia entre dos clases si cambios en la definición de una puede afectar el funcionamiento de la otra.

Sean A y B dos clases, ejemplos de dependencias incluyen:

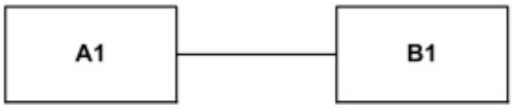
- A le manda un mensaje a B.
- A define una operación que incluye como parámetro a uno del tipo de B.
- A crea una instancia de B.



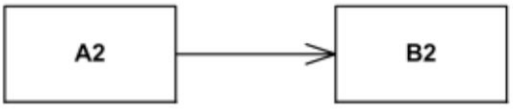
Asociaciones

Las asociaciones son relaciones de dependencia más fuertes. Implican que una clase contiene como referencia a una instancia de otra.

Pueden incluir una etiqueta que describa la semántica de la asociación, una multiplicidad a ambos lados de la relación e indicadores de navegabilidad.

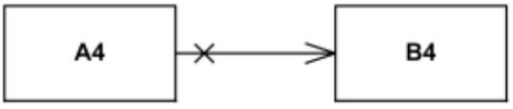


Both ends of association have ***unspecified navigability****.*

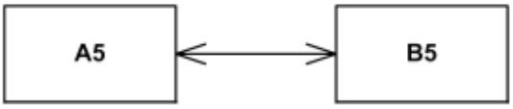


A2 has ***unspecified navigability*** *while B2 is* ***navigable*** *from A2.*

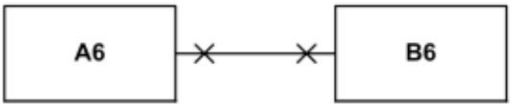
A3 is ***not navigable*** *from B3 while B3 has* ***unspecified navigability****.*



A4 is ***not navigable*** *from B4 while B4 is* ***navigable*** *from A4.*



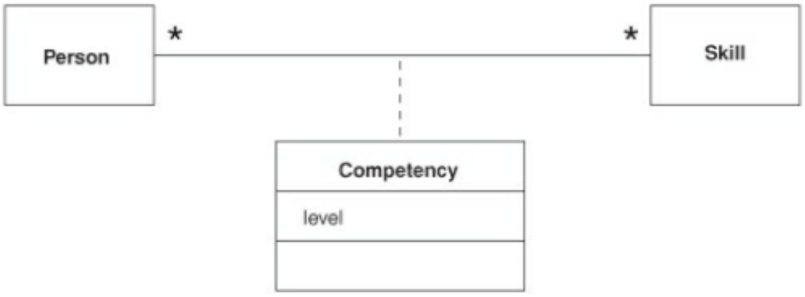
A5 is ***navigable*** *from B5 and B5 is* ***navigable*** *from A5.*



A6 is ***not navigable*** *from B6 and B6 is* ***not navigable*** *from A6.*

Clase de asociación

A veces, una asociación misma puede tener características que desee capturar en atributos. UML permite asignar una clase a una asociación.



Asociación calificada

Permite ilustrar el concepto de Hash o Diccionario.

En el ejemplo anterior, ojo con la multiplicidad. En este ejemplo, lo que estoy ilustrando es que "Order" tiene un HashMap del tipo Product -> OrderLine. Dado un "Product" en particular, puedo acceder a 0 o 1 "OrderLine" (esto es lo que me dice la multiplicidad). Independiente de lo anterior, "Order" obviamente tiene N "OrderLine" asociados.

Agregación

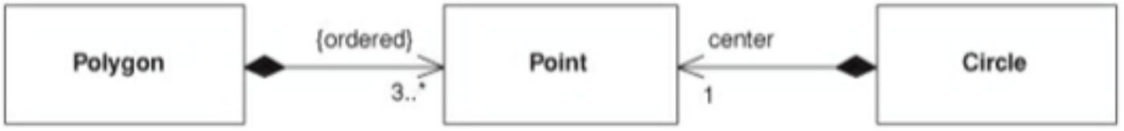
Una agregación es una asociación en donde se explicita que la instancia referenciada por la source puede también ser referenciada por alguien más. Se le asocia una semántica de: [Source] "tiene" [Target].
Ejemplos:

- Un empleado tiene una oficina asignada, pero nada impide que otras personas (clientes, visitantes, internos) tengan la misma oficina.
- Un curso tiene estudiantes, pero nada impide que otras instancias (talleres, becas, diplomados) tengan al mismo estudiante.

Composición

Una composición es una asociación en donde se explicita que la instancia referenciada por la source no puede estar siendo referenciada por nadie más. Se le asocia una semántica de: [Target] “es parte intrínseca de” [Source].
Ejemplo:

- Una página es parte intrínseca de un libro; dada una página de un libro, ningún otro libro puede tener la misma página simultáneamente.
- Un motor es parte intrínseca de un auto; dado un motor de un auto, ningún otro vehículo puede tener el mismo motor simultáneamente.

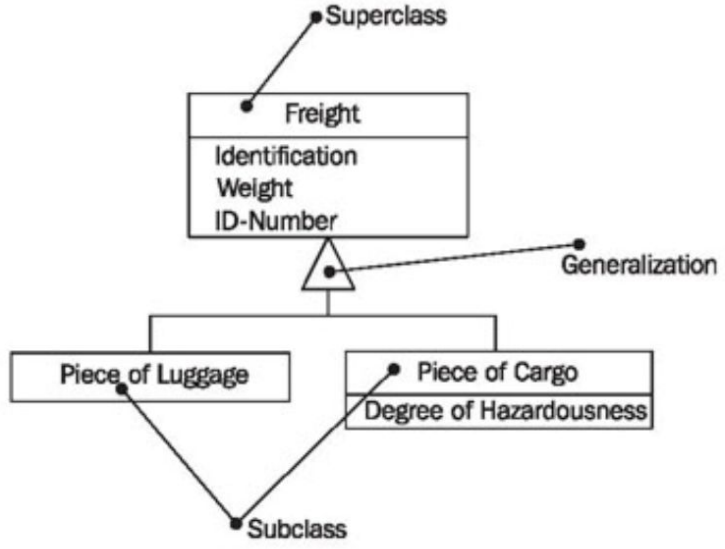


Agregación vs Composición

	Asociación	
	Agregación	**Composición**
Vida	propia	la del dueño
Relación	tiene	es parte de
Ejemplo	Auto tiene Conductor	Motor es parte de Auto

Generalización

Permite establecer similitudes entres múltiples clases. En lenguajes de programación, es análogo al concepto de "herencia" entre clases. Se establece una semántica de: [Target] "es un/una" [Source].



Se recomienda solo usar generalización entre clases si se cumple la relación semántica is-a. Evitar usar generalización entre clases con el único propósito de reutilizar código. Evitar, en la medida de lo posible, jerarquías profundas de clases.

Interfaces

Las interfaces deben marcarse con la etiqueta <<interface>>. Clases que implementan interfaces se representan mediante una línea punteada; interfaces que extienden otras se representan mediante una línea sólida.