TEMA 4: DISEÑO ORIENTADO A OBJETOS Y TEMA 5: OPTIMIZACIÓN Y DOCUMENTACIÓN

Entornos de Desarrollos

Antonio Salces Alcaraz (1º DAM)
C.P.I.F.P. Alan Turing
12/02/2025

Índice

TEMA 4: DISEÑO ORIENTADO A OBJETOS

1. Introducción

El desarrollo de software es un proceso que pretende solucionar problemas utilizando herramientas informáticas, resultando así la construcción de un programa. Debemos de realizar un análisis, una especificación del proceso, y otra de los resultados esperados.

2. Notación de los diagramas de clase

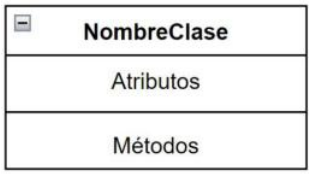
2.1. Clases, atributos, métodos y visibilidad

2.1.1. Clase

Define de manera genérica como van a ser los objetos de un determinado tipo.

Una clase por sí sola no tiene sentido, ya que es un concepto. Para poder usarla, debemos de instanciarla, es decir, crear un objeto de esa clase.

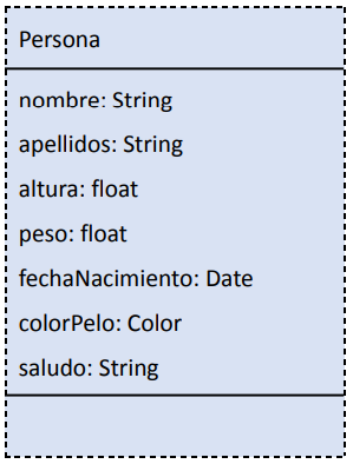
En UML, una clase se representa de la siguiente manera:



2.1.2. Atributo

Son las propiedades que posee una clase. Cada uno tiene un **nombre** y un **tipo**.

Los atributos en UML se representan en la parte central del rectángulo.



Los tipos de datos utilizados dependen del lenguaje utilizado para codificar, aunque UML ofrece los suyos propios: tipos básicos, clases definidas por usuario, tipos o colecciones genéricos, y tipos numerados.

Tipos básicos

Tipos primitivos utilizados normalmente en la mayoría de lenguajes (int, double, float, char, String, boolean…).

Clases definidas por el usuario

Cualquier clase definida puede ser usada como tipo para atributo o parámetros (Cliente, Persona…).

Tipos de colecciones o genéricos

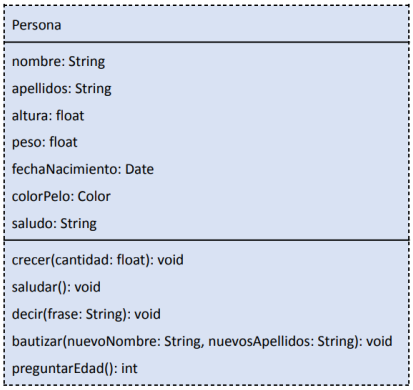
Utilizados para representar estructuras de datos con múltiples elementos (List<T>, Map<T>…).

Tipos enumerados

Tipo especial que contiene un conjunto limitado de posibles valores (Color: rojo, verde o azul).

2.1.3. Método

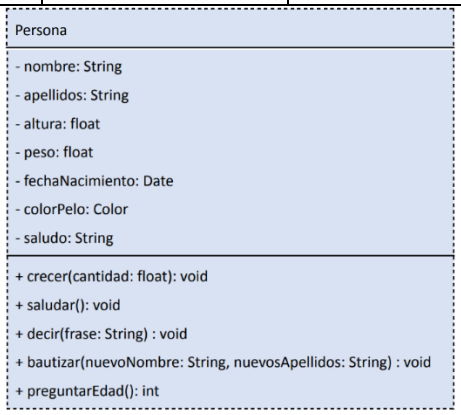
Funciones u operaciones que las clases u objetos pueden realizar, llamado **comportamiento de la clase**. Las funciones en UML se representan de la siguiente forma, con los parámetros entre paréntesis.



2.1.4. Visibilidad

En UML, los **modificadores de acceso** determinan la visibilidad de atributos y métodos de una clase.

Modificador de acceso	Símbolo	Descripción
Público	+	Visible desde cualquier clase.
Privado	-	Visible solo dentro de la propia clase.
Protegido	#	Visible dentro de la clase y sus subclases.
Paquete (por defecto)	~	Visible solo dentro del mismo paquete.



2.2. Objetos. Instanciación

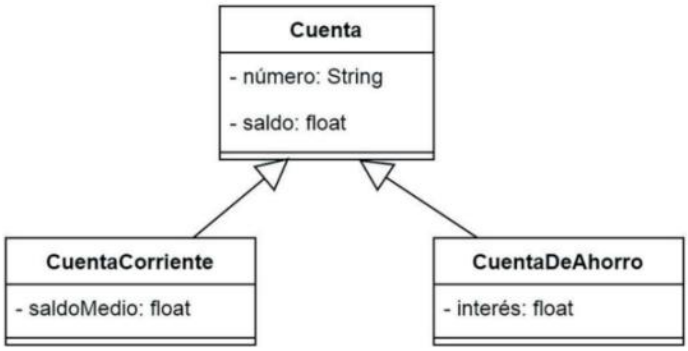
Para instanciar un objeto, debemos de usar la palabra clave "**new**". Ejemplo:

Clase o tipo de variable	Nombre de la variable	Operador de asignación	Palabra clave para instanciar	Llamada al constructor
Persona	persona1	=	new	Persona();

2.3. Relaciones. Herencia, composición, agregación, asociación y uso

2.3.1. Herencia

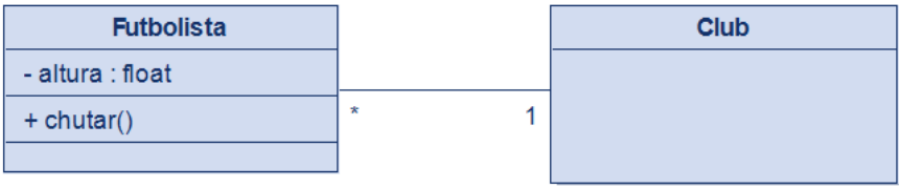
Se establece jerarquía de clases y subclases con métodos y atributos comunes. La superclase declara los atributos y métodos, y las subclases los heredan, añadiendo los propios de la subclase. Ejemplo en UML:



Se representa uniendo mediante una flecha de la subclase a la superclase. **La punta será un triángulo hueco**.

2.3.2. Asociación

Las clases pretenden representar conceptos del mundo real, y no suelen aparecer solas, sino que están relacionadas con otras.



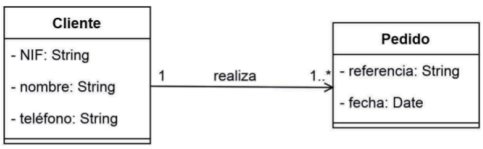
La cardinalidad significa que un futbolista concreto puede pertenecer a un club, mientras que un club puede tener a varios jugadores. Cardinalidades existentes:

- 1: uno.
- 0..1: cero o uno.
- 0..*: cero o varios, también representado como "*".
- 1..*: uno o varios.
- N: el número indicado.
- N..M: entre los números N y M.

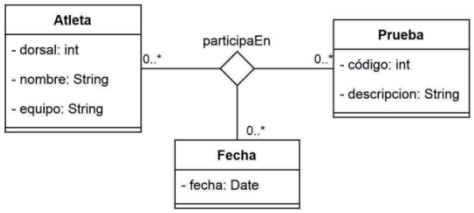
Las asociaciones pueden ser de distinto tipo dependiendo de cuantas clases vinculan, recibiendo el nombre de **grado de la relación**.

- Reflexivas: de grado 1, vincular una clase consigo misma.
- Binarias: de grado 2, vinculan dos clases.
- Ternarias, cuaternarias…: grado 3, 4…, vincular tres, cuatro… clases, respectivamente.

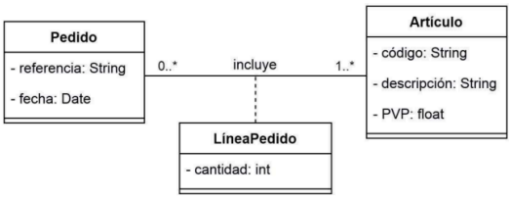
Para las asociaciones, agregaciones y composiciones, podemos indicar su **navegabilidad**, siendo esto el sentido en el que se puede acceder a la información desde una clase de una asociación. Se representa mediante una flecha que indica en qué sentido es navegable.



En las relaciones de más de grado 2, la relación se representa con un rombo, que une las clases.



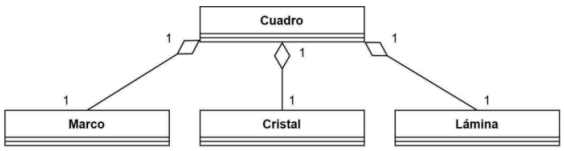
En algunas ocasiones, necesitaremos almacenar alguna información relevante sobre las asociaciones. Para ello, crearemos una clase asociativa.



2.3.3. Agregación (o agregación débil)

Representa una clase que está compuesta por otras. Existen los componentes y el compuesto.

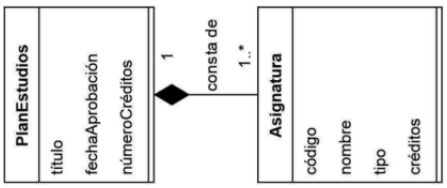
El tiempo de vida de los componentes es distinto al del compuesto, es decir, los componentes podían y pueden existir sin que haya un compuesto. También implica que debe haber mínimo un componente para que exista un compuesto.



2.3.4. Composición (o agregación fuerte)

Tipo especial de agregación en la que la multiplicidad del lado del compuesto es siempre 1, y en que los componentes dependen del compuesto. Tiene dos restricciones:

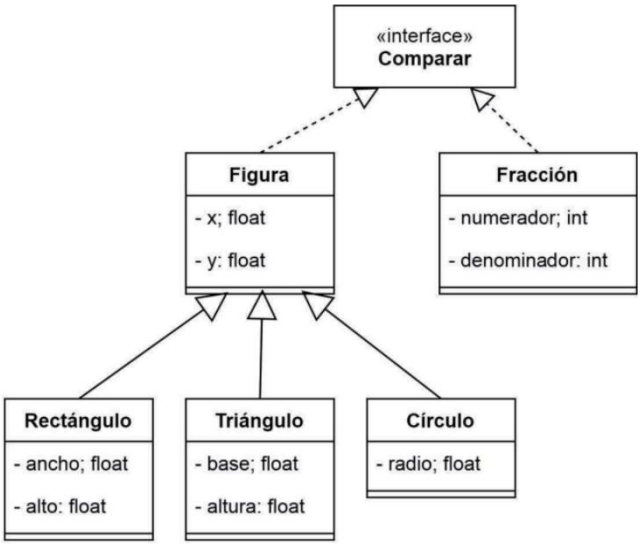
- Cada componente solo puede estar presente en un compuesto.
- Si se elimina el compuesto, hay que eliminar a todos los componentes.



2.3.5. Realización. Implementación de interfaces

Una interfaz es un conjunto de operaciones que especifican un servicio de una clase o componente. Una interfaz de representa gráficamente mediante un rectángulo que incluye su nombre con la leyenda <<interface>> y sus operaciones.

Una relación de realización es la que hay entre una interfaz y las clases que la implementan.



En este diagrama se puede ver cómo hay tres subclases, que heredan de una misma superclase, y esta implementa la misma interfaz que otra clase.

2.3.6. Clases abstractas

Se pone el nombre de la clase en cursiva. Incluye al menos un método abstracto, que también se pone en cursiva.

2.3.7. Atributos y métodos estáticos

Se indican con subrayado.

3. Herramientas para la elaboración de diagramas de clase. Instalación

Algunas de las herramientas más conocidas para realizar diagramas UML son:

- Magic Draw.
- Visual Paradigm.
- Modelio.
- StarUML.
- UMLet.
- diagrams.net

TEMA 5: OPTIMIZACIÓN Y DOCUMENTACIÓN

4. Patrones de refactorización

4.1. Patrones de refactorización

Nombre	Name	Descripción
Renombrar	Rename	Cambia el nombre de una clase, interfaz, variable o método a algo más significativo y actualiza todo el código fuente del proyecto para reflejar este cambio.
Mover clase	Move class	Mueve una clase a otro paquete o dentro de una clase. Además, todo el código fuente del proyecto es actualizado para hacer referencia a la clase en el nuevo paquete.
Copiar clase	Copy class	Copia una clase en el mismo paquete o en una diferente.
Eliminar de forma segura	Safe delete	Elimina, asegurándose de que no van a quedar referencias perdidas al código eliminado.
Descender	Push down	Mueve clases, métodos y campos internos a una subclase de la clase actual
Encapsular campos	Encapsulate field	Genera métodos get y set para un campo y, opcionalmente, actualiza todas las referencias a este campo usando los métodos get y set.
Cambiar firma de método	Change method signature	Agregar, eliminar, modificar o cambiar el orden de los parámetros de un método o cambiar el modificador de acceso (público, privado, protegido). Cambiar el nombre del método.
Pasar del nivel interior al exterior	Move inner to outer level	Mueve una clase miembros hacia un nivel exterior. No confundir con la herencia. Se refiere a clases internar, que pueden heredar de la externa o no.
Hacer estático	Make static	Transforma en estático un método.
Migración de tipo	Type migration	Cambia el tipo de un atributo o método
Extraer interfaz	Extract interface	Crea una nueva interfaz formada a partir del método público no estático seleccionado en una clase o interfaz.

4.2. Generador de código

Nombre	Name	Descipción
Constructor	Constructor	Crea un constructor y permite elegir los parámetros que recibe.
Getters	Getters	Crea los getters de los atributos que se seleccionen.
Setters	Setters	Crea los setterts de los atributos que se seleccionen.
Getters y setters	Getters y setters	Crea los getters y setters de los atributos que se seleccionen.
Sobrescribir métodos	Override methods	Sobrescribe los métodos que se seleccionen. Permite elegir los métodos de todas las clases de las que hereda.
toString()	toString()	Sobrescribe el método toString y permite seleccionar los atributos.
Test	Test	Permite generar una clase para realizar test unitarios.