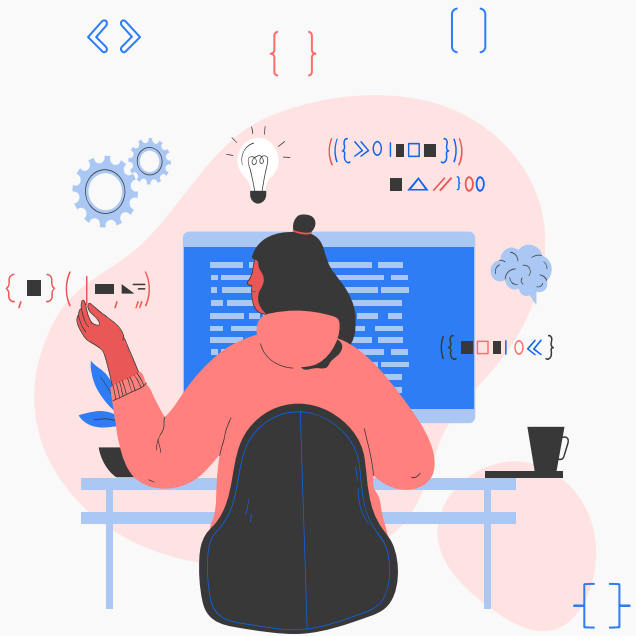
Entornos de desarrollo

Tema 5 – UML. Diagramas de clase

Marina Hurtado Rosales
marina.hurtado@escuelaartegranada.com



Índice

Programación Orientada a Objetos

Programación estructurada

En la **programación estructurada**:

- El problema se divide en **subproblemas**
- Cada subproblema se resuelve mediante **funciones**
- El **programa** se construye **encadenando funciones**

Este enfoque:

- Está **centrado en las acciones**
- Es **difícil de mantener** en programas grandes
- Presenta **poca reutilización** del código

Programación Orientada a Objetos

La programación orientada a objetos (POO) propone un enfoque diferente:

- El problema se modela mediante **objetos**
- Cada objeto representa una **entidad** del mundo real
- Los objetos combinan **datos** y **comportamiento**

Este paradigma facilita:

- Programas más claros
- Mejor mantenimiento
- Mayor reutilización

Objetos: atributos y métodos

Un objeto está formado por:

Atributos

- Son los datos que describen al objeto
- Representan su estado interno

Métodos

- Definen el comportamiento del objeto
- Modifican o consultan sus atributos

Cuando se solicita a un objeto que ejecute un método, se dice que se le envía un mensaje.

Principios POO: Abstracción

La **abstracción** consiste en

- Identificar las **características comunes** de varios objetos
- Ignorar los detalles innecesarios
- Crear una **representación simplificada** del problema

Es un concepto clave para:

- Analizar problemas
- Diseñar clases
- Modelar sistemas complejos

Principios POO: Encapsulación

La **encapsulación** implica:

- **Agrupar** atributos y métodos relacionados
- **Ocultar** los detalles internos de una clase
- **Controlar el acceso** al estado del objeto

Gracias a la encapsulación:

- Se protege la información
- Se reduce la complejidad
- Se mejora la mantenibilidad

Principios POO: Modularidad

La **modularidad** permite:

- **Dividir** una aplicación en partes independientes
- Desarrollar cada parte por separado
- Facilitar la **reutilización** y el **mantenimiento**

En POO, los **objetos** son los **módulos básicos** del sistema.

Principios POO: Polimorfismo y herencia

Polimorfismo

- Un mismo **método** puede **comportarse de forma diferente**
- Depende del objeto que lo ejecute

Herencia

- Una clase puede **reutilizar atributos y métodos** de otra
- Permite crear **jerarquías de clases**

Ventajas de la POO

La programación orientada a objetos ofrece:

- Reutilización de código
- Reducción del tiempo de desarrollo
- Facilidad de mantenimiento
- Mayor escalabilidad
- Mejor adaptación a cambios

Interfaz e implementación de una clase

Una clase se puede entender en dos partes:

Interfaz

- Qué puede hacer la clase
- Métodos accesibles desde fuera

Implementación

- Cómo lo hace
- Código interno y detalles

Esta separación mejora la claridad del diseño.

Visibilidad de elementos de una clase

La visibilidad controla el acceso a los elementos de una clase. Podemos encontrar principalmente cuatro tipos de visibilidad:

- **Pública**: el elemento es accesible desde cualquier clase
- **Privada**: el elemento es sólo accesible desde la propia clase
- **Protegida**: el elemento es accesible desde las clases derivadas
- **Paquete**: el elemento es accesible desde los archivos del mismo paquete

Normas de visibilidad

Como norma general en POO:

- Los **atributos** deben ser **privados**
- Los **métodos principales** deben ser **públicos**
- Los **métodos auxiliares** pueden ser **privados** o **protegidos**

Estas normas refuerzan la encapsulación.

Instanciación

La instanciación es el proceso de crear un objeto a partir de una clase.

- La clase define la estructura
- El objeto es una instancia real en memoria
- En Java, se produce al llamar al constructor con la palabra reservada **new**

Características de un objeto

Un objeto se define por:

- **Estado**: valores de sus atributos
- **Comportamiento**: métodos públicos
- **Tiempo de vida**: desde su creación hasta su destrucción

UML: Unified Modeling Language

¿Qué es UML?

UML (Unified Modeling Language) es un lenguaje estándar que:

- Permite **modelar** sistemas software
- Facilita el **diseño** de aplicaciones orientadas a objetos
- **Documenta** proyectos de forma clara
- Es **independiente** del lenguaje de programación.

¿Por qué modelar?

Modelar un sistema permite:

- Comprender mejor el problema
- Comunicar ideas dentro del equipo
- Detectar errores antes de programar
- Documentar el diseño del sistema

Elementos de UML

Un diagrama UML se compone de:

- **Elementos estructurales**: son los nodos del grafo. Definen el tipo de diagrama (clases, objetos, componentes...)
- **Relaciones**: son los arcos del grafo, que establecen conexiones entre elementos
- **Notas**: aclaraciones que permiten escribir comentarios que nos ayuden a entender algún concepto que queramos representar
- **Agrupaciones**: organización por paquetes. Facilitan el desarrollo de sistemas grandes

Tipos de diagramas UML

Tipos de diagramas UML

En UML existen diferentes tipos de diagramas, pero principalmente se agrupan en dos tipos:

Diagramas estructurales: representan la visión estática del sistema

- Diagramas de clases
- Diagrama de despliegue
- Diagramas de objetos
- Diagramas de componentes
- Diagrama de estructura

Diagramas de comportamiento: muestran la conducta en tiempo de ejecución del sistema

- Diagrama de actividad
- Diagrama de máquina de estados
- Diagrama de casos de uso
- Diagramas de interacción: diagramas de tiempos, diagramas de secuencia, diagrama de comunicación...

Diagrama de clases

Diagrama de clases

El diagrama de clases:

- Representa la estructura estática del sistema
- Muestra clases y relaciones
- Es el más importante de UML
- Se utiliza como plano previo al código.

Representación de una clase

Una clase se representa mediante un rectángulo dividido en:

- **Nombre de la clase**
- **Atributos**
- **Métodos**

Atributos en UML

Los atributos se representan indicando:

Visibilidad

- Privada –
- Publica +
- Protegida #
- Paquete ~

Nombre

Tipo de dato (int, double, boolean…)

```
- nombre : String
- edad : int
```

Métodos en UML

Los métodos incluyen:

Visibilidad

- Privada –
- Publica +
- Protegida #
- Paquete ~

Nombre

Parámetros (junto con su tipo)

Tipo de retorno (int, double, boolean...)

```
+ saludar() : void
```

Elementos del diagrama de clases

Además de las clases, los diagramas están formados por los siguientes elementos:

- **Notas:** Se utilizan para aclarar ciertas partes del diagrama que pueden ser confusas

- **Paquetes:** Se representan con rectángulos y se utilizan para agrupar las clases que están recogidas en el mismo paquete

Relaciones entre clases

En los diagramas de clases existen distintos tipos de relaciones:

- **Asociación** -> relación simple
- **Agregación** -> relación todo-parte débil
- **Composición** -> relación todo-parte fuerte
- **Herencia** -> relación “es un”

Relaciones: asociación

La asociación es una relación débil entre clases en la que:

- Una clase conoce a otra. Se almacena sólo una parte de la información de la clase.

- No hay dependencia de existencia.

- Puede ser **bidireccional**, cuando ambas clases de la relación guardan información de la otra: Se indica con una línea continua sin flechas.

- También puede ser **unidireccional**, cuando se guarda sólo la información de la clase que indica la flecha. Se representa con una línea con una flecha.

Ejemplo: Alumno — Curso

Del enunciado al diagrama

Análisis de un enunciado

Para transformar un texto en un diagrama

- Leer el enunciado **completo**. Subrayar los **verbos** y los **sustantivos**.
- Identificar **sustantivos**, que pueden ser **clases** o **atributos**. Si el sustantivo tiene **entidad propia**, es una **clase**, si es una **característica** de un elemento, es un **atributo**.
- Identificar **acciones** que puede hacer la entidad. Estas acciones serán los **métodos.**
- Detectar **relaciones** entre clases. Una vez detectadas, analizar de qué tipo son.

Ejemplo práctico

En un centro educativo se quiere desarrollar una aplicación para gestionar alumnos y cursos.

De cada alumno se desea almacenar su nombre, edad y número de expediente. Un alumno puede matricularse en un curso y consultar sus datos personales.

De cada curso se quiere guardar su nombre, el código del curso y el número máximo de alumnos. Un curso puede añadir alumnos y mostrar la información del curso.

Ejemplo práctico

Paso 1: subrayamos todos los sustantivos y ponemos en cursiva todos los verbos.

En un centro educativo se quiere ***desarrollar*** una aplicación para ***gestionar*** alumnos y cursos.

De cada alumno se desea ***almacenar*** su nombre, edad y número de expediente. Un alumno puede ***matricularse*** en un curso y ***consultar*** sus datos personales.

De cada curso se quiere ***guardar*** su nombre, el código del curso y el número máximo de alumnos. Un curso puede ***añadir*** alumnos y ***mostrar*** la información del curso.

Ejemplo práctico

Paso 2: analizamos lo que tenemos subrayado y señalamos únicamente aquello que es importante.

En un centro educativo se quiere ***desarrollar*** una aplicación para ***gestionar*** alumnos y cursos.

De cada alumno se desea ***almacenar*** su nombre, edad y número de expediente. Un alumno puede ***matricularse*** en un curso y ***consultar*** sus datos personales.

De cada curso se quiere ***guardar*** su nombre, el código del curso y el número máximo de alumnos. Un curso puede ***añadir*** alumnos y ***mostrar*** la información del curso.

Ejemplo práctico

Paso 3: quitamos aquellos sustantivos, verbos o conceptos que estén duplicados.

En un centro educativo se quiere ***desarrollar*** una aplicación para ***gestionar*** alumnos y cursos.

De cada alumno se desea ***almacenar*** su nombre, edad y número de expediente. Un alumno puede ***matricularse*** en un curso y ***consultar*** sus datos personales.

De cada curso se quiere ***guardar*** su nombre, el código del curso y el número máximo de alumnos. Un curso puede ***añadir*** alumnos y ***mostrar*** la información del curso.

Ejemplo práctico

Paso 4: analizamos los sustantivos y conceptos que nos quedan, y señalamos con un color diferente aquellos que representan a entidades y aquellos que son características.

En un centro educativo se quiere ***desarrollar*** una aplicación para ***gestionar*** alumnos y cursos.

De cada alumno se desea ***almacenar*** su nombre, edad y número de expediente. Un alumno puede ***matricularse*** en un curso y ***consultar*** sus datos personales.

De cada curso se quiere ***guardar*** su nombre, el código del curso y el número máximo de alumnos. Un curso puede ***añadir*** alumnos y ***mostrar*** la información del curso.

Ejemplo práctico

Paso 5: analizamos los verbos que nos quedan subrayados, y retiramos aquellos que estén relacionados con el almacenaje o guardado de características.

En un centro educativo se quiere ***desarrollar*** una aplicación para ***gestionar*** alumnos y cursos.

De cada alumno se desea ***almacenar*** su nombre, edad y número de expediente. Un alumno puede ***matricularse*** en un curso y ***consultar*** sus datos personales.

De cada curso se quiere ***guardar*** su nombre, el código del curso y el número máximo de alumnos. Un curso puede ***añadir*** alumnos y ***mostrar*** la información del curso.

Ejemplo práctico

Con esto ya tendríamos el análisis inicial. Las entidades se corresponderían con las **clases**, las características con los **atributos**, y los *verbos* con los ***métodos***.

De esta manera quedaría algo de este estilo:

Alumno: nombre, edad, numeroExpediente.

- *matricularseEnCurso()*
- *consultarDatos()*

Curso: nombre, código, numMaxAlumnos.

- *añadirAlumno()*
- *mostrarInformacion()*

En un centro educativo se quiere ***desarrollar*** una aplicación para ***gestionar*** alumnos y cursos.

De cada alumno se desea ***almacenar*** su nombre, edad y número de expediente. Un alumno puede ***matricularse*** en un curso y ***consultar*** sus datos personales.

De cada curso se quiere ***guardar*** su nombre, el código del curso y el número máximo de alumnos. Un curso puede ***añadir*** alumnos y ***mostrar*** la información del curso.

Ejemplo práctico

Paso 6: Una vez que tenemos este análisis inicial, volvemos a mirar el enunciado y buscamos frases que relacionen a dos o más clases y las señalamos. Estas frases indican la aparición de **relaciones** entre las clases.

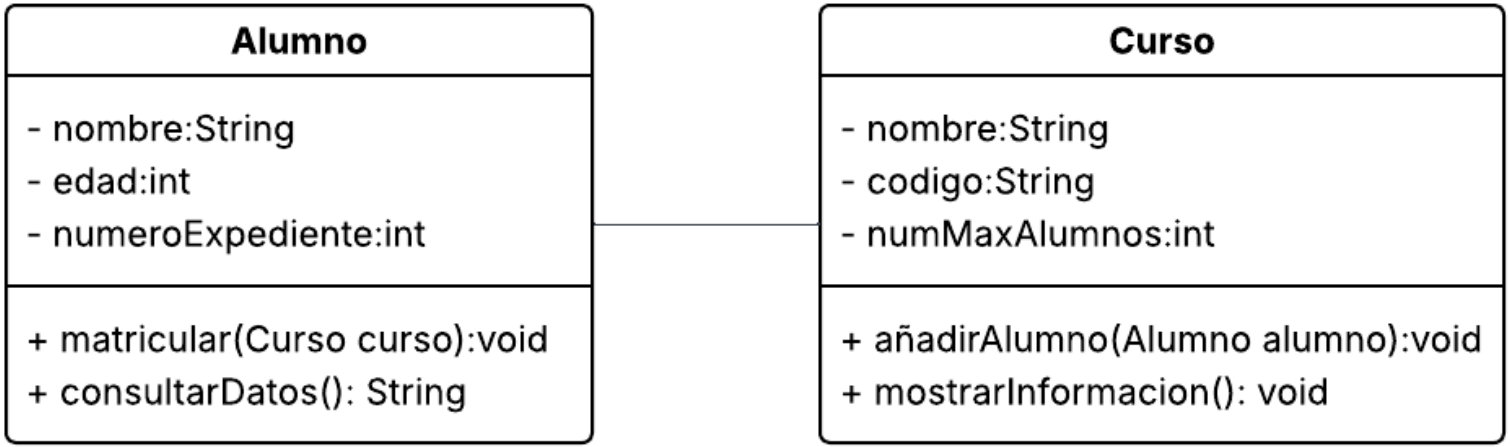
En este caso hay una frase que relaciona alumnos con cursos, y otra que relaciona cursos con alumnos. Sin embargo, ninguna clase depende de la otra para existir, por lo que tendríamos una **relación de asociación bidireccional** entre ambas clases

En un centro educativo se quiere desarrollar una aplicación para gestionar alumnos y cursos.

De cada alumno se desea almacenar su nombre, edad y número de expediente. Un alumno puede matricularse en un curso y consultar sus datos personales.

De cada curso se quiere guardar su nombre, el código del curso y el número máximo de alumnos. Un curso puede añadir alumnos y mostrar la información del curso.

Ejemplo práctico

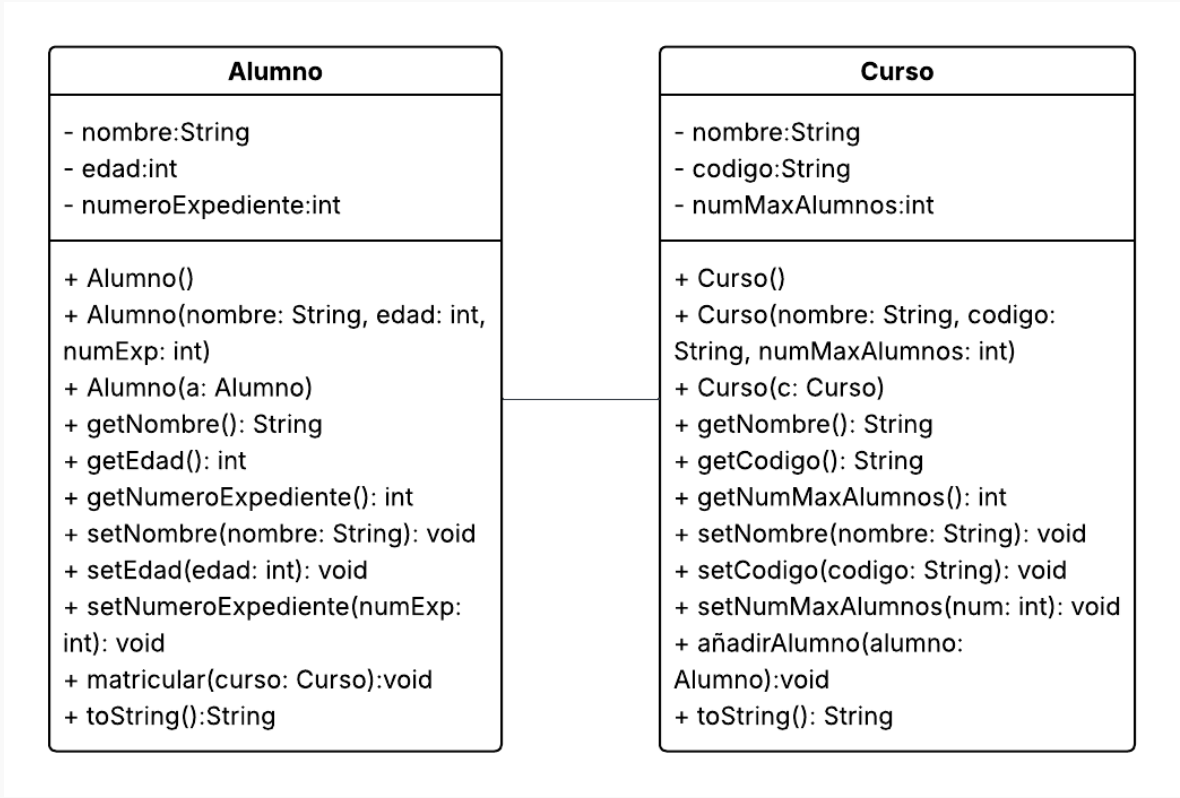


Ejemplo práctico

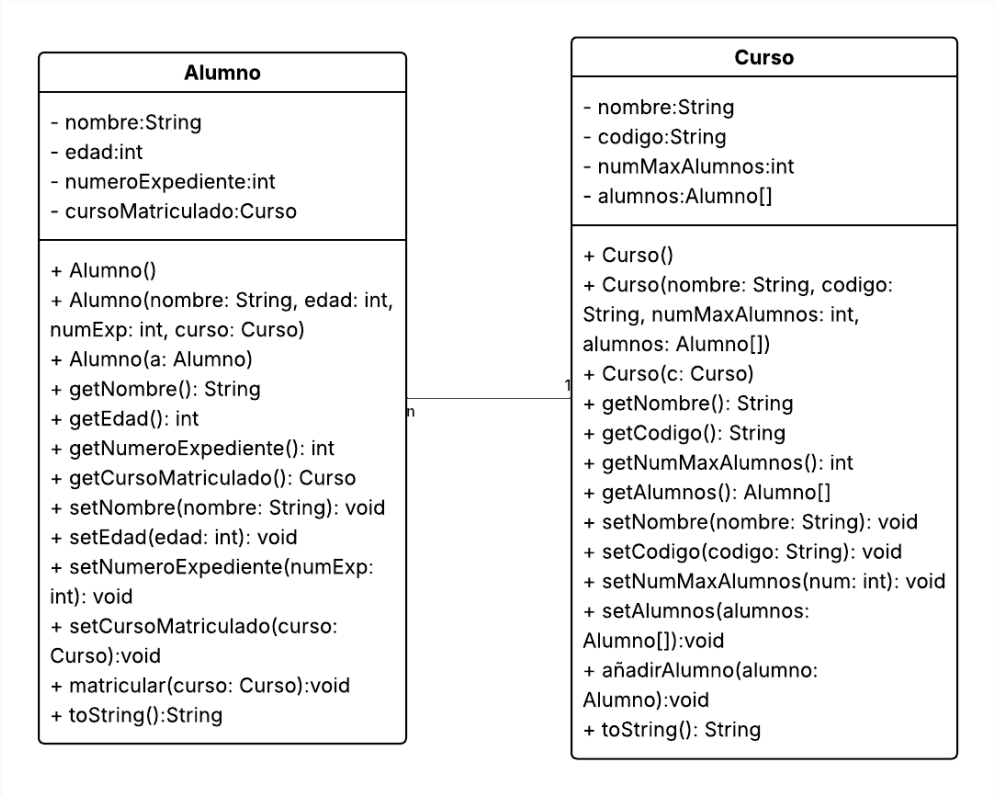
Este diagrama, a pesar de ser correcto y adecuado al enunciado, está incompleto y le faltan métodos. Hay que añadir todos los métodos que son obligatorios en una clase: constructores, getters, setters, toString... Además de esto, hay que analizar si algunos de los métodos que tenemos serán redundantes y hay que quitarlos.

En este caso, el método consultarDatos de Alumno sería redundante cuando tengamos los getters, y el método mostrarInformación de Curso sería redundante cuando tengamos el toString.

Ejemplo práctico



Ejemplo práctico



Del diagrama a código

Correspondencia UML a Java

Una vez tenemos el diagrama de clases refinado, podemos usarlo de base para empezar a programar. Cada uno de los elementos que tenemos en el diagrama se corresponden con un elemento en concreto en el código de Java:

- Clase UML → class en Java
- Atributos UML → atributos de la clase, variables en Java
- Método → métodos de la clase, funciones de Java
- Visibilidad → public (+) / private (-) / protected (#) / por defecto (~)

Herramientas UML

Herramientas para diagramas UML

Existen diferentes herramientas que se pueden utilizar para dibujar distintos tipos de diagramas UML, tanto web como locales.

Draw.io
- Gratuita
- Sin registro
- Versión online y de escritorio
- Soporta los diagramas UML

Lucidchart
- Gratuita
- Requiere registro
- Soporte UML
- Sencilla
- Aplicación web

StarUML
- Aplicación de escritorio
- Más cercana al entorno profesional

Modelio
- Aplicación de escritorio
- Cercana al entorno profesional

Visual Paradigm
- Aplicación de escritorio
- Cercana al entorno profesional
- Usada en educación superior, sobre todo en la Universidad