
ANALYZE
DEVELOPMENT
DESIGN
IMPLEMENT

5.1. Diagramas estructurales

5.1.2 Diagramas de clases

En un diagrama de clases podemos encontrar los siguientes elementos:

- ✓ **Clases**: recordemos que son abstracciones del dominio del sistema que representan elementos de éste mediante una serie de **características, que llamaremos atributos, y su comportamiento, que serán métodos.** Los atributos y métodos tendrán una visibilidad que determinará quien puede acceder al atributo o método. Por ejemplo, una clase puede representar a un coche, sus atributos serán la cilindrada, la potencia y la velocidad, y tendrá dos métodos, uno para acelerar, que subirá la velocidad, y otro para frenar que la bajará.
- ✓ **Relaciones**: en el diagrama representan **relaciones reales entre los elementos del sistema a los que hacen referencia las clases.** Pueden ser de **asociación, agregación y herencia.** Por ejemplo, si tengo una clase persona, puedo establecer una relación conduce entre persona y coche.
- ✓ **Notas**: Se representan como un cuadro donde podemos escribir comentarios que nos ayuden a entender algún concepto que queramos representar.
- ✓ **Elementos de agrupación**: Se utilizan cuando hay que modelar un sistema grande, entonces las clases y sus relaciones **se agrupan en paquetes**, que a su vez se relacionan entre sí.

5.1.2.1. Creación de clases.

Una clase se representa en el diagrama como un rectángulo divido en tres filas, arriba aparece el nombre de la clase, a continuación, los atributos con su visibilidad y después los métodos con su visibilidad que está representada por el signo menos "–" para los atributos (privados) y por el signo más "+" para los métodos (públicos).

"Una clase es una descripción de un conjunto de objetos que manifiestan los mismos atributos, operaciones, relaciones y la misma semántica."
(Object Modelling and Design [Rumbaught et al., 1991])

"Una clase es un conjunto de objetos que comparten una estructura y un comportamiento comunes."
[Booch G., 1994]

5.1.2.2 Atributos.

Forman la parte estática de la clase. Son un conjunto de variables para las que es preciso definir:

- ✓ Su nombre.
- ✓ Su tipo, puede ser un tipo simple, que coincidirá con el tipo de dato que se seleccione en el lenguaje de programación final a usar, o compuesto, pudiendo incluir otra clase.

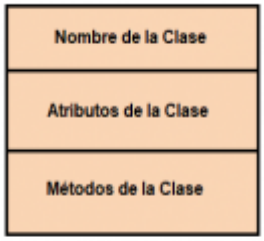
Además, se pueden indicar otros datos como un valor inicial o su visibilidad. La visibilidad de un atributo se puede definir como:

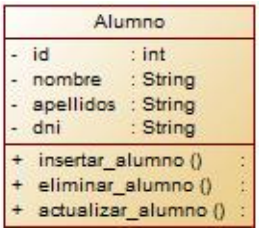
- ✓ **Público**: Se pueden acceder desde cualquier clase y cualquier parte del programa.
- ✓ **Privado**: Sólo se pueden acceder desde operaciones de la clase.
- ✓ **Protegido**: Sólo se pueden acceder desde operaciones de la clase o de clases derivadas en cualquier nivel.
- ✓ **Paquete**: Se puede acceder desde las operaciones de las clases que pertenecen al mismo paquete que la clase que estamos definiendo. Se usa cuando el lenguaje de implementación es Java.

5.1.2.3. Métodos.

Representan la funcionalidad de la clase, es decir, qué puede hacer. Para definir un método hay que indicar como mínimo su **nombre, parámetros, el tipo que devuelve y su visibilidad.** También se debe incluir una descripción del método que aparecerá en la documentación que se genere del proyecto.

Existen un caso particular de método, el **constructor** de la clase, que tiene como característica que no devuelve ningún valor. El constructor tiene el mismo nombre de la clase y **se usa para ejecutar las acciones necesarias cuando se instancia un objeto de la clase.** Cuando haya que destruir el objeto se podrá utilizar una función para ejecutar las operaciones necesarias cuando el objeto deje de existir, que dependerán del lenguaje que se utilice.





3.4. Relaciones entre clases.

Una **relación** es **una conexión entre dos clases** que incluimos en el diagrama cuando aparece algún tipo de relación entre ellas en el dominio del problema.

Se representan como una línea continua. Los mensajes "navegan" por las relaciones entre clases, es decir, los mensajes se envían entre objetos de clases relacionadas, normalmente en ambas direcciones, aunque a veces la definición del problema hace necesario que se navegue en una sola dirección, entonces la línea finaliza en punta de flecha.

Las relaciones se caracterizan por su cardinalidad, que representa cuantos objetos de una clase se pueden involucrar en la relación, y pueden ser:

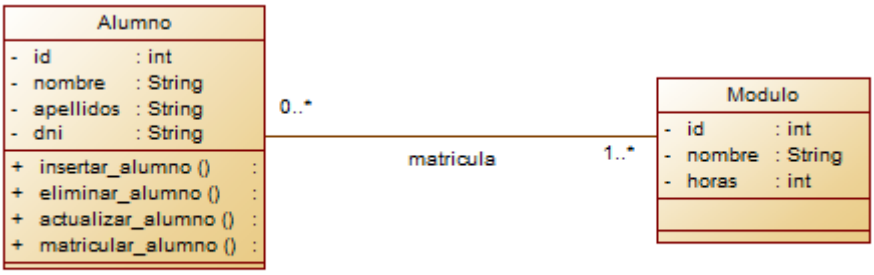
- ✓ De herencia.
- ✓ De composición.
- ✓ De agregación.

5.1.2.4. Cardinalidad o multiplicidad de la relación

Un concepto muy importante es la **cardinalidad de una relación**, **representa cuantos objetos de una clase se van a relacionar con objetos de otra clase.** En una relación hay dos cardinalidades, una para cada extremo de la relación y pueden tener los siguientes valores:

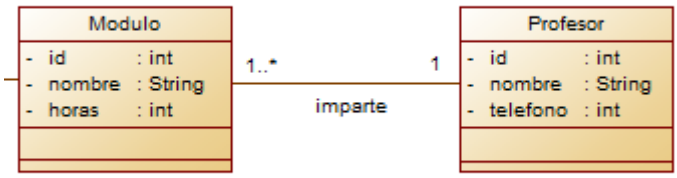
Significado de las cardinalidades.	
Cardinalidad	**Significado**
1	Uno y sólo uno
0..1	Cero o uno
N..M	Desde N hasta M
*	Cero o varios
0..*	Cero o varios
1..*	Uno o varios (al menos uno)

Por ejemplo, si tengo la siguiente relación:



Quiere decir que los alumnos se matriculan en los módulos, en concreto, que un alumno se puede matricular en uno a más módulos y que un módulo puede tener ningún alumno, uno o varios.

O esta otra:



Donde un profesor puede impartir uno o varios módulos, mientras que un módulo es impartido sólo por un profesor.

3.4.2. Relación de herencia.

La **herencia** es una propiedad que **permite a los objetos ser construidos a partir de otros objetos**, es decir, la capacidad de un objeto para utilizar estructuras de datos y métodos presentes en sus antepasados.

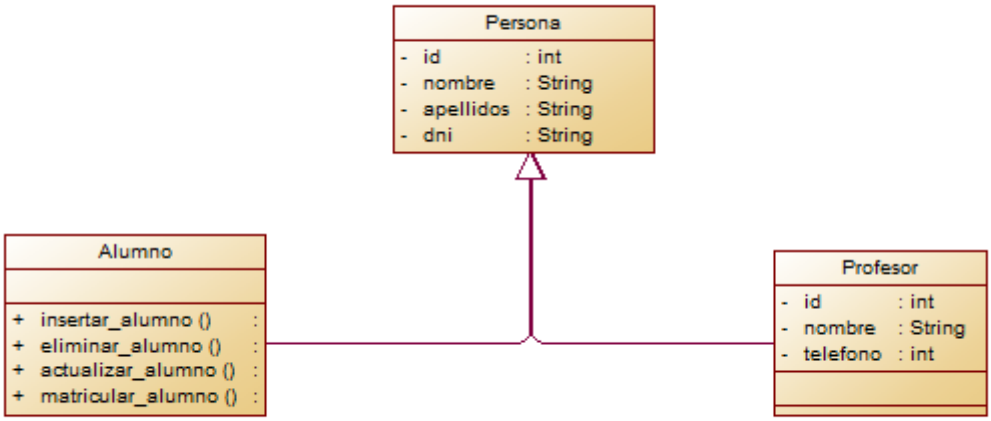
El objetivo principal de la herencia es la ***reutilización***, poder utilizar código desarrollado con anterioridad. La herencia supone una clase base y una jerarquía de clases que contiene las clases derivadas. Las clases derivadas pueden heredar el código y los datos de su clase base, añadiendo su propio código especial y datos, incluso cambiar aquellos elementos de la clase base que necesitan ser diferentes, es por esto por lo que los atributos, métodos y relaciones de una clase se muestran en el nivel más alto de la jerarquía en el que son aplicables.

Tipos:

- ✓ **Herencia simple**: Una clase puede tener sólo un ascendente. Es decir, una subclase puede heredar datos y métodos de una única clase base.
- ✓ **Herencia múltiple**: Una clase puede tener más de un ascendente inmediato, adquirir datos y métodos de más de una clase.

Representación:

En el diagrama de clases se representa como una asociación en la que el extremo de la clase base tiene un triángulo.



3.4.3. Agregación y composición.

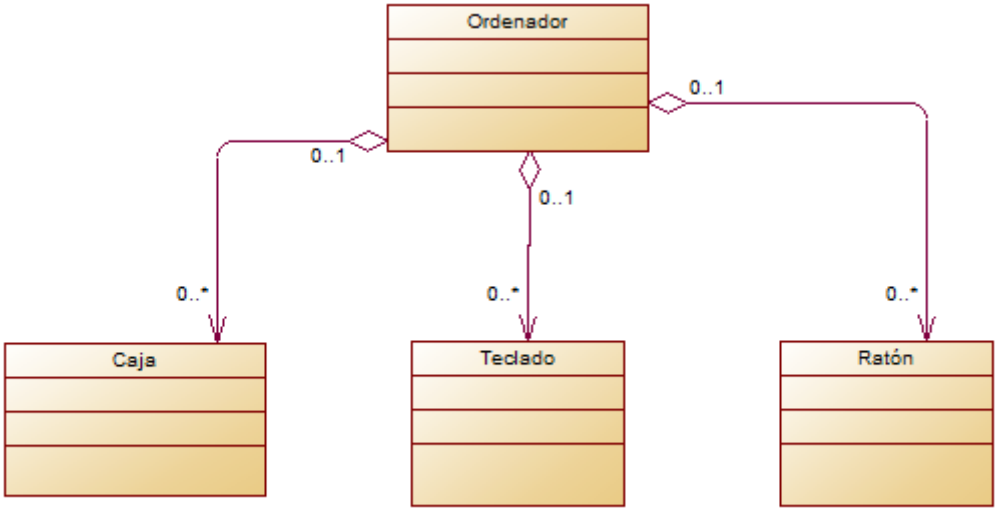
Muchas veces **una determinada entidad existe como un conjunto de otras entidades.** En este tipo de relaciones **un objeto componente se integra en un objeto compuesto.** La orientación a objetos recoge este tipo de relaciones como dos conceptos: **la agregación y la composición.**

La **agregación** es una asociación binaria que **representa una relación todo-parte** (pertenece a, tiene un, es parte de). **Los elementos parte pueden existir sin el elemento contenedor** y no son propiedad suya. Por ejemplo, un centro comercial tiene clientes o un equipo tiene unos miembros. El tiempo de vida de los objetos no tiene por qué coincidir.

En el siguiente caso, tenemos un ordenador que se compone de piezas sueltas que pueden ser sustituidas y que tienen entidad por sí mismas, por lo que se representa mediante relaciones de agregación. Utilizamos la agregación porque es posible que una caja, ratón o teclado o una memoria RAM existan con independencia de que pertenezcan a un ordenador o no.

Representación:

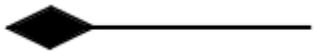
En el diagrama de clases se representa como una asociación en la que el extremo de la clase contenedora tiene un rombo blanco.



La **composición** es **una agregación fuerte en la que una instancia 'parte' está relacionada, como máximo, con una instancia 'todo' en un momento dado**, de forma que cuando un objeto 'todo' es eliminado, también son eliminados sus objetos 'parte'. Por ejemplo: un rectángulo tiene cuatro vértices, un centro comercial está organizado mediante un conjunto de secciones de venta...
Para modelar la estructura de un ciclo formativo vamos a usar las clases Módulo, Competencia y Ciclo que representan lo que se puede estudiar en Formación Profesional y su estructura lógica. Un ciclo formativo se compone de una serie de competencias que se le acreditan cuando supera uno o varios módulos formativos.

Representación:

En el diagrama de clases se representa como una asociación en la que el extremo de la clase contenedora tiene un rombo negro.



Modulo Competencia Ciclo

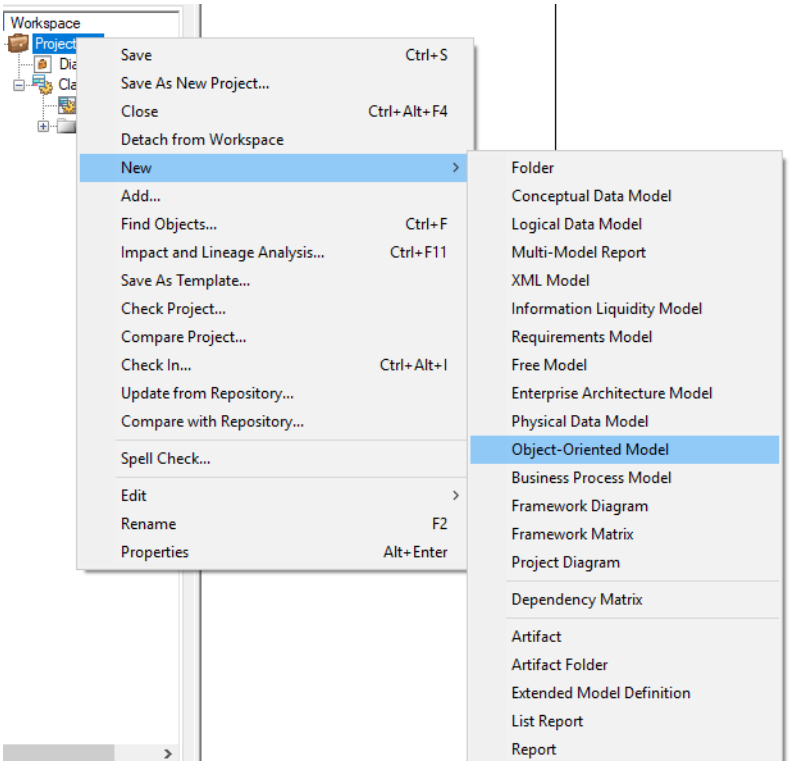
Dado que si eliminamos el ciclo las competencias no tienen sentido, y lo mismo ocurre con los módulos hemos usado relaciones de composición. Si los módulos o competencias pudieran seguir existiendo sin su contenedor habríamos utilizado relaciones de agregación.

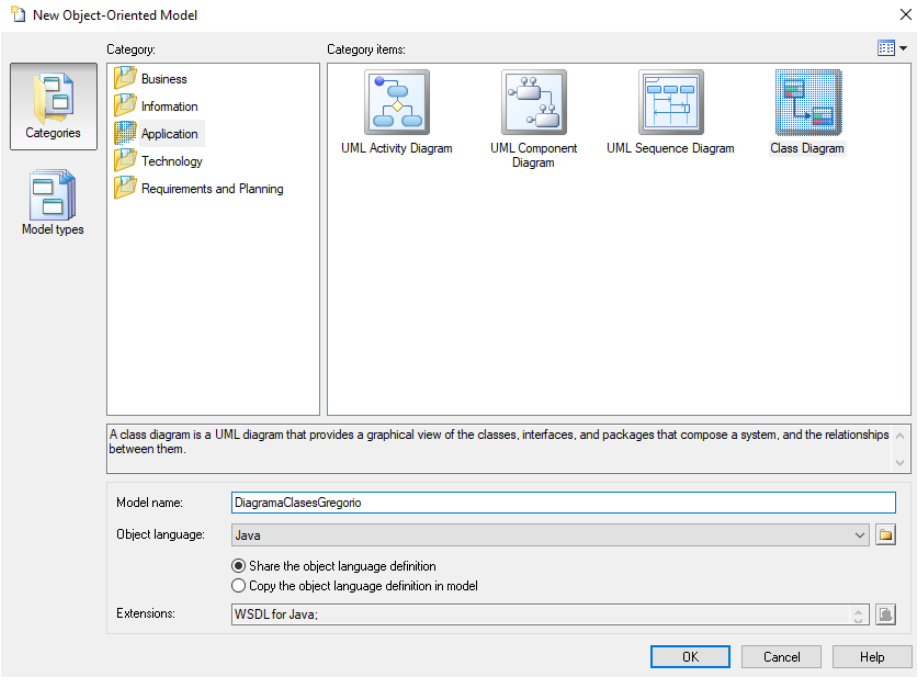
Estas relaciones se representan con un rombo en el extremo de la entidad contenedora. En el caso de la agregación es de color blanco y para la composición negro. Como en toda relación hay que indicar la cardinalidad.

Creación de nuestro primer Diagrama dentro de *Power Designer:*

Añadimos un nuevo diagrama a nuestro proyecto, hacemos click derecho y vamos a:

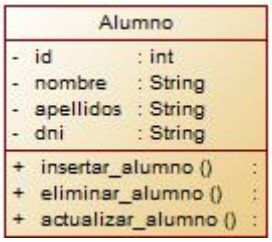
New->Object-Oriented Model



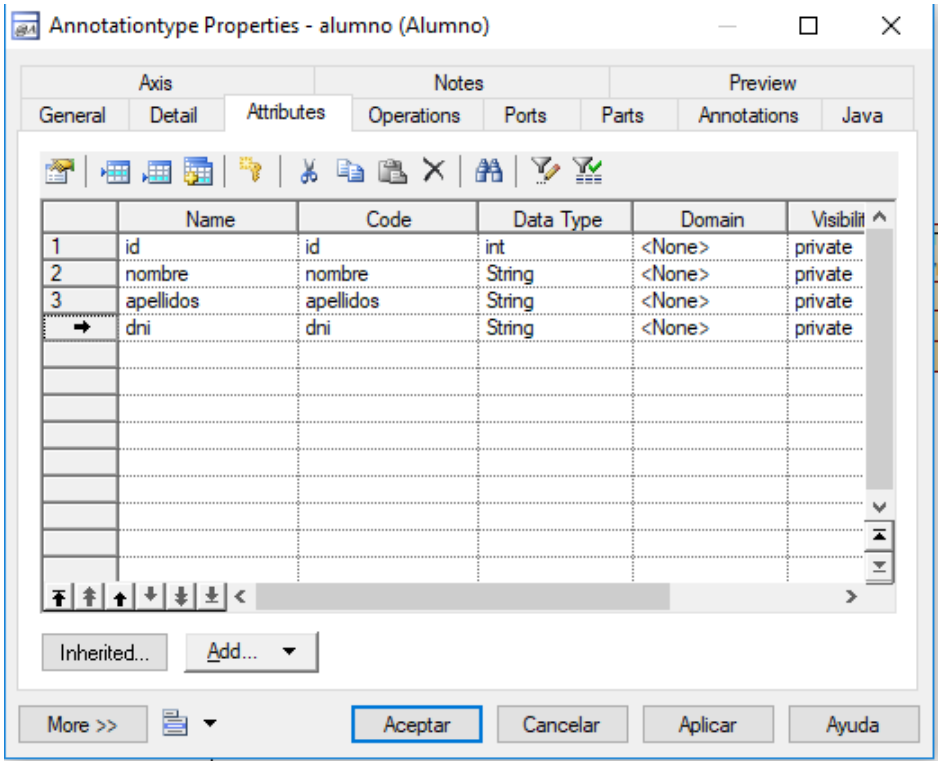


Elegimos la opción de '*Class Diagram*'

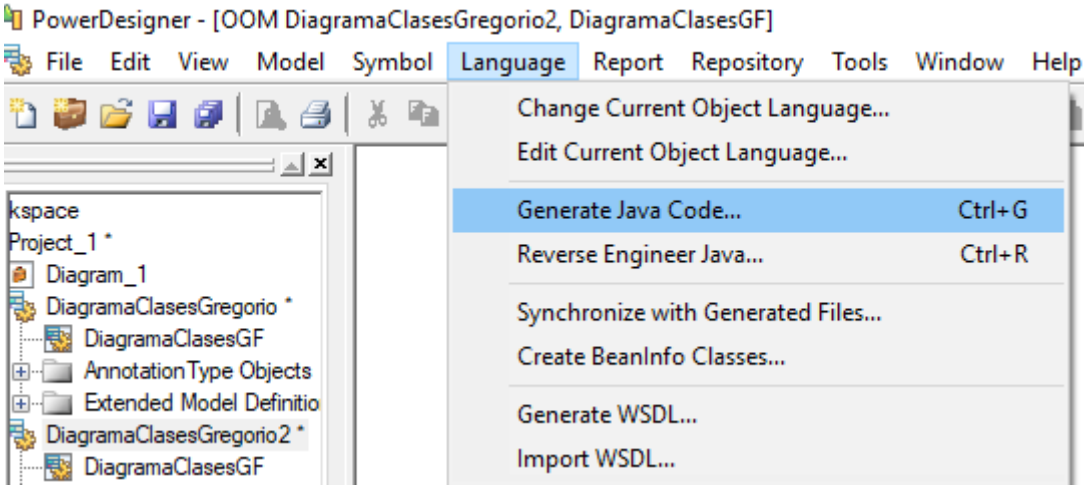
Insertamos una clase 'alumno' dentro de nuestro diagrama, dentro de la cual configuraremos algunos atributos y métodos.



Accedemos a las propiedades de nuestra clase, donde configuraremos sus métodos, atributos…



Generaremos el resto de las clases para nuestro diagrama, y las relaciones entre ellos. Llegados a este punto, tendremos la opción de generar el código Java a partir del diagrama generado, o de incluso a partir de un código Java, generar el diagrama asociado, mediante **Ingeniería Inversa**.



En la siguiente captura podemos ver la Paleta de *Power Designer* para la inserción de clases, relaciones entre clases, agregaciones, composiciones, herencias…