

Diagramas de clases UML

Autor: Fernando Berzal

Adaptación: Antonio H.

Representación gráfica de una clase (notación UML)

Una clase se representa con un rectángulo dividido en tres partes:

- El nombre de la clase
(identifica la clase de forma unívoca)
- Sus atributos
(datos asociados a los objetos de la clase)
- Sus operaciones
(comportamiento de los objetos de esa clase)

IMPORTANTE: Las clases se deben identificar con un nombre que, por lo general, pertenecerá al vocabulario utilizado habitualmente al hablar del problema que tratamos de resolver.

Representación de una clase de la manera más simple:

```
public class Cuenta
{
    ...
}
```

Cuenta

Representación de atributos y operaciones de una clase:

```
public class Cuenta
{
    private double balance = 0;
    private double limit;

    public void ingresar (...) ...
    public void retirar (...) ...
}
```

Cuenta
-balance -límite
+ingresar() +retirar()

Especificación completa de la clase:

Cuenta
-balance : Dinero = 0 -límite : Dinero
+ingresar(in cantidad : Dinero) +retirar(in cantidad : Dinero)

```
public class Cuenta
{
    private double balance = 0;
    private double limit;

    public void ingresar (double cantidad)
    {
        balance = balance + cantidad;
    }

    public void retirar (double cantidad)
    {
        balance = balance - cantidad;
    }
}
```

Clase de ejemplo

Representación gráfica en UML:

Motocicleta
-matrícula -color -velocidad -en_marcha
+arrancar() +acelerar() +frenar() +girar()

Definición en Java:

Fichero `Motocicleta.java`

```
public class Motocicleta
{
    // Atributos (variables de instancia)
    private String matricula; // Placa de matrícula
    private String color;      // Color de la pintura
    private int velocidad;     // Velocidad actual (km/h)
    private boolean en_marcha; // ¿moto arrancada?

    // Operaciones (métodos)

    public void arrancar ()
    { ... }
    public void acelerar ()
    { ... }
    public void frenar ()
    { ... }
    public void girar (float angulo)
    { ... }
}
```

Relaciones entre clases:

Las relaciones existentes entre las distintas clases nos indican cómo se comunican los objetos de esas clases entre sí:

Los mensajes “navegan”
por las relaciones existentes entre las distintas clases.

Existen distintos tipos de relaciones:

- **Asociación** (conexión entre clases)
- **Pertenencia** (agregación y composición)
- **Dependencia** (relación de uso)
- **Generalización/especialización** (relaciones de herencia)

Asociación

Una asociación es una relación estructural que describe una conexión entre objetos.

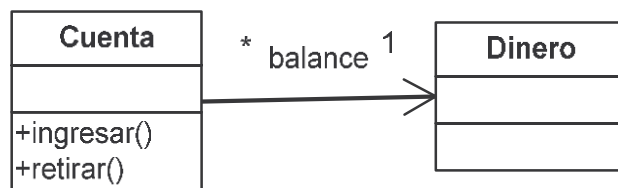


Gráficamente, se muestra como una línea continua que une las clases relacionadas entre sí.

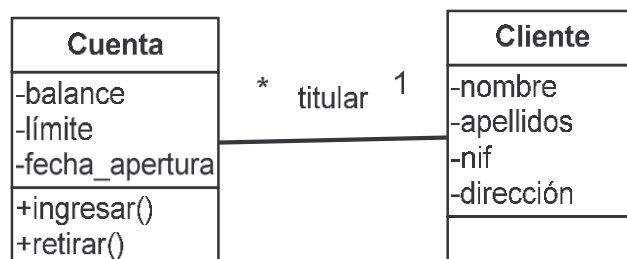
Navegación de las asociaciones

Aunque las asociaciones suelen ser bidireccionales (se pueden recorrer en ambos sentidos), en ocasiones es deseable hacerlas unidireccionales (restringir su navegación en un único sentido).

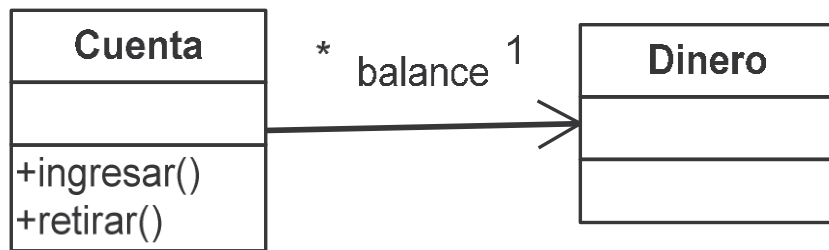
Gráficamente, cuando la asociación es unidireccional, la línea termina en una punta de flecha que indica el sentido de la asociación:



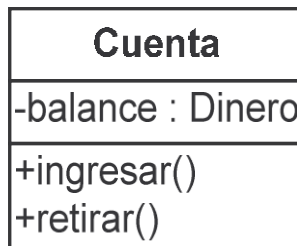
Asociación unidireccional



Asociación bidireccional



equivale a



```

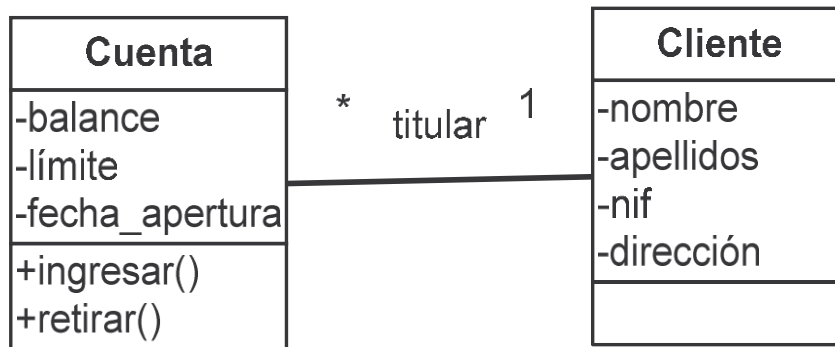
class Cuenta
{
    private Dinero balance;

    public void ingresar (Dinero cantidad)
    {
        balance += cantidad;
    }

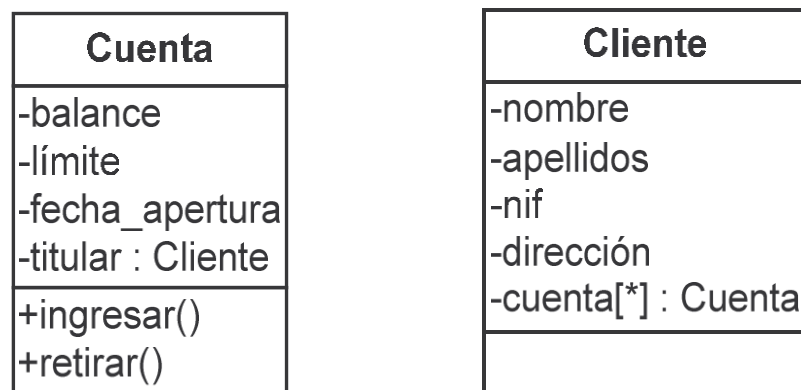
    public void retirar (Dinero cantidad)
    {
        balance -= cantidad;
    }

    public Dinero getSaldo ()
    {
        return balance;
    }
}
  
```

Hemos supuesto que `Dinero` es un tipo de dato con el que se pueden hacer operaciones aritméticas y hemos añadido un método adicional que nos permite comprobar el saldo de una cuenta.



viene a ser lo mismo que



con la salvedad de
que el enlace bidireccional hemos de mantenerlo nosotros

```

public class Cuenta
{
    ...
    private Cliente titular;
    ...
}

public class Cliente
{
    ...
    private Cuenta cuenta[];
    ...
}
  
```

Un cliente puede tener varias cuentas, por lo que en la clase cliente hemos de mantener un conjunto de cuentas (un vector en este caso).

Multiplicidad de las asociaciones

La multiplicidad de una asociación determina cuántos objetos de cada tipo intervienen en la relación:

El número de instancias de una clase que se relacionan con UNA instancia de la otra clase.

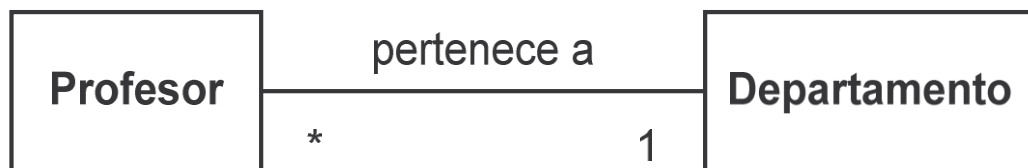
- Cada asociación tiene dos multiplicidades (una para cada extremo de la relación).
- Para especificar la multiplicidad de una asociación hay que indicar la multiplicidad mínima y la multiplicidad máxima (mínima..máxima)

Multiplicidad	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)

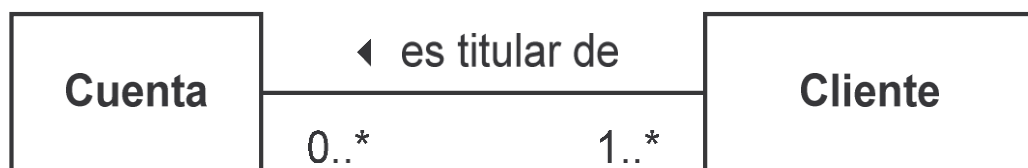
- Cuando la multiplicidad mínima es 0, la relación es opcional.
- Una multiplicidad mínima mayor o igual que 1 establece una relación obligatoria.



Todo departamento tiene un director.
Un profesor puede dirigir un departamento.



Todo profesor pertenece a un departamento.
A un departamento pueden pertenecer varios profesores.



Relación opcional
Un cliente puede o no
ser titular de una cuenta

Relación obligatoria
Una cuenta ha de tener
un titular como mínimo

Relaciones involutivas

Cuando la misma clase aparece en los dos extremos de la asociación.



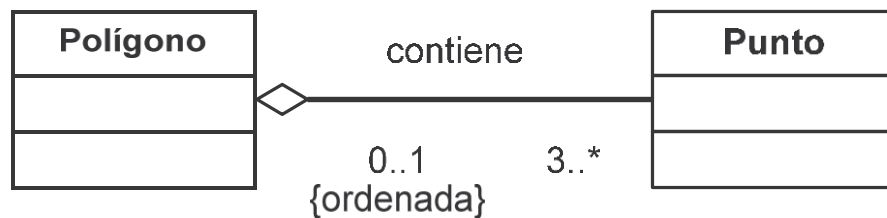
Agregación y composición

Casos particulares de asociaciones:
Relación entre un todo y sus partes

Gráficamente,
se muestran como asociaciones con un rombo en uno de los extremos.

Agregación

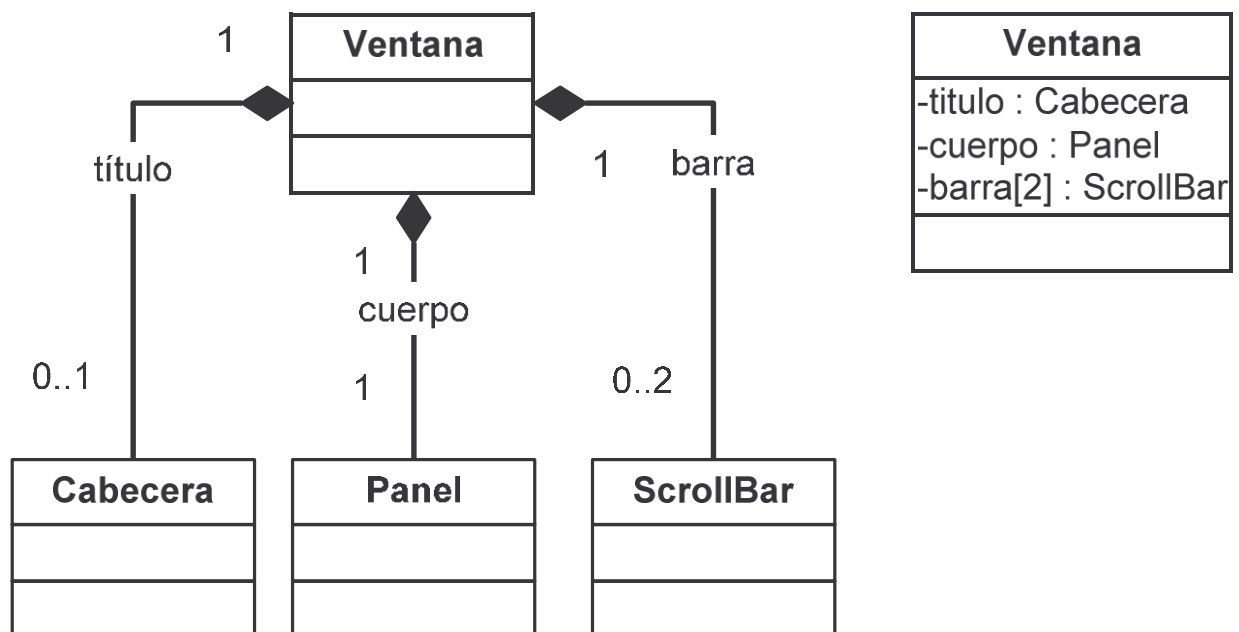
Las partes pueden formar parte de distintos agregados.



Composición

Agregación disjunta y estricta:

Las partes sólo existen asociadas al compuesto
(sólo se accede a ellas a través del compuesto)



Dependencia

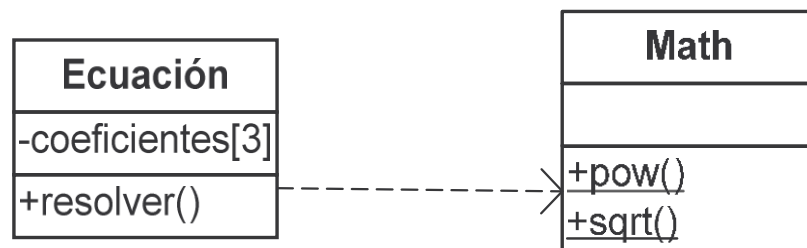
Relación (más débil que una asociación) que muestra la relación entre un cliente y el proveedor de un servicio usado por el cliente.

- Cliente es el objeto que solicita un servicio.
- Servidor es el objeto que provee el servicio solicitado.

Gráficamente, la dependencia se muestra como una línea discontinua con una punta de flecha que apunta del cliente al proveedor.

Ejemplo

Resolución de una ecuación de segundo grado



$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Para resolver una ecuación de segundo grado hemos de recurrir a la función `sqrt` de la clase `Math` para calcular una raíz cuadrada.

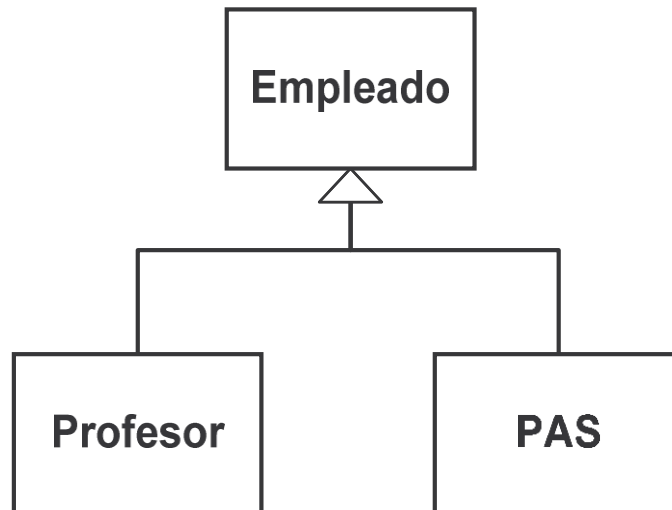
NOTA:

La clase `Math` es una clase “degenerada” que no tiene estado. Es, simplemente, una colección de funciones de cálculo matemático.

Herencia (generalización y especialización)

La relación entre una superclase y sus subclasses

Objetos de distintas clases pueden tener atributos similares y exhibir comportamientos parecidos (p.ej. animales, mamíferos...).



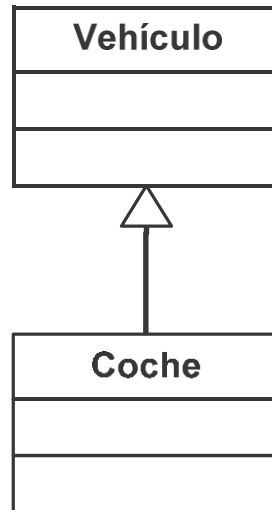
```
public class Empleado
{
    ...
}

public class Profesor extends Empleado
{
    ...
}

public class PAS extends Empleado
{
    ...
}
```

La noción de clase está próxima a la de conjunto:

Generalización y especialización
expresan relaciones de inclusión entre conjuntos.



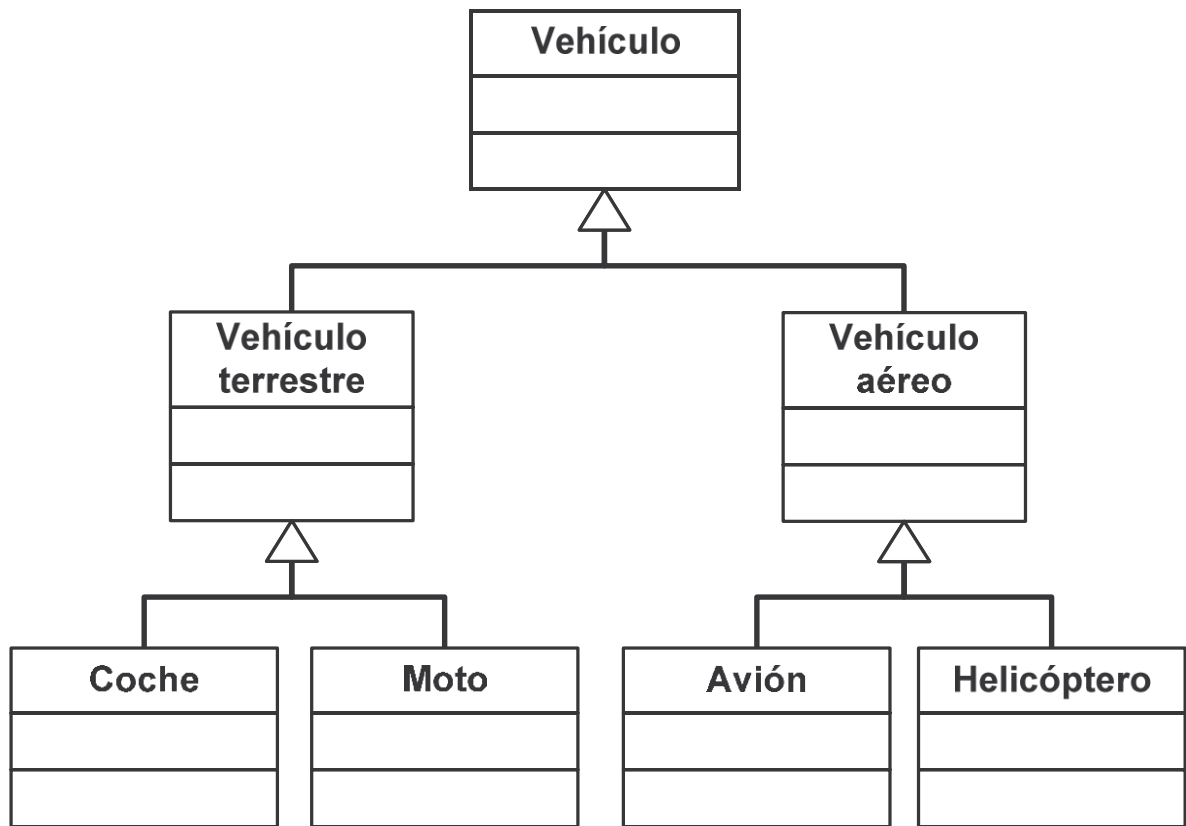
Instancias: **coches** \subset **vehículos**

- Todo coche es un vehículo.
- Algunos vehículos son coches.

Propiedades: **propiedades(coches)** \supset **propiedades(vehículos)**

- Un coche tiene todas las propiedades de un vehículo.
- Algunas propiedades del coche no las tienen todos los vehículos.

Jerarquías de clases



Las clases se organizan en una estructura jerárquica formando una taxonomía.

- El comportamiento de una categoría más general es aplicable a una categoría particular.
- Las subclases heredan características de las clases de las que se derivan y añaden características específicas que las diferencian.

En el diagrama de clases, los atributos, métodos y relaciones de una clase se muestran en el nivel más alto de la jerarquía en el que son aplicables.

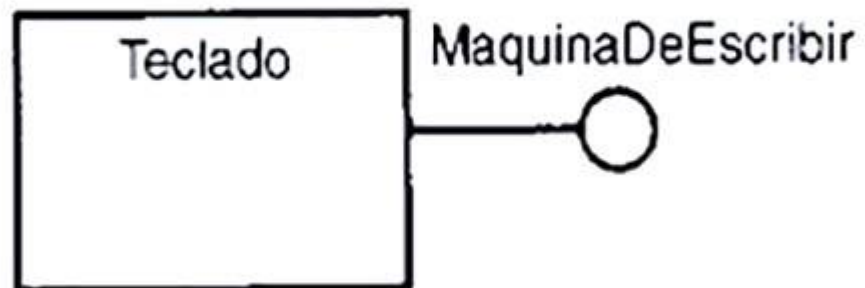
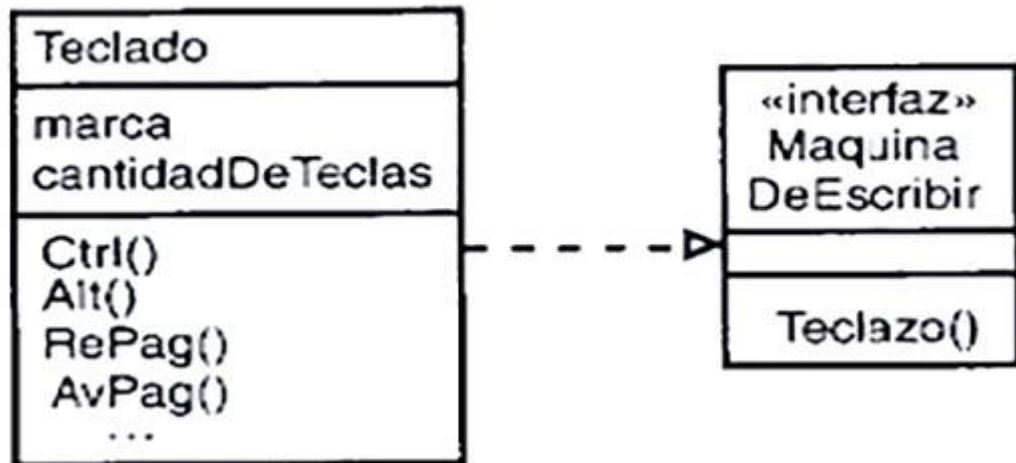
Visibilidad de los miembros de una clase

Se pueden establecer distintos niveles de encapsulación para los miembros de una clase (atributos y operaciones) en función de desde dónde queremos que se pueda acceder a ellos:







Visibilidad	Significado	Java	UML
Pública	Se puede acceder al miembro de la clase desde cualquier lugar.	<code>public</code>	+
Protegida	Sólo se puede acceder al miembro de la clase desde la propia clase o desde una clase que herede de ella.	<code>protected</code>	#
Paquete o friendly	Sólo se puede acceder al miembro de la clase desde el mismo paquete		~
Privada	Sólo se puede acceder al miembro de la clase desde la propia clase.	<code>private</code>	-

Interfaces en UML

Hay dos maneras de representarlas, con más o menos detalle:



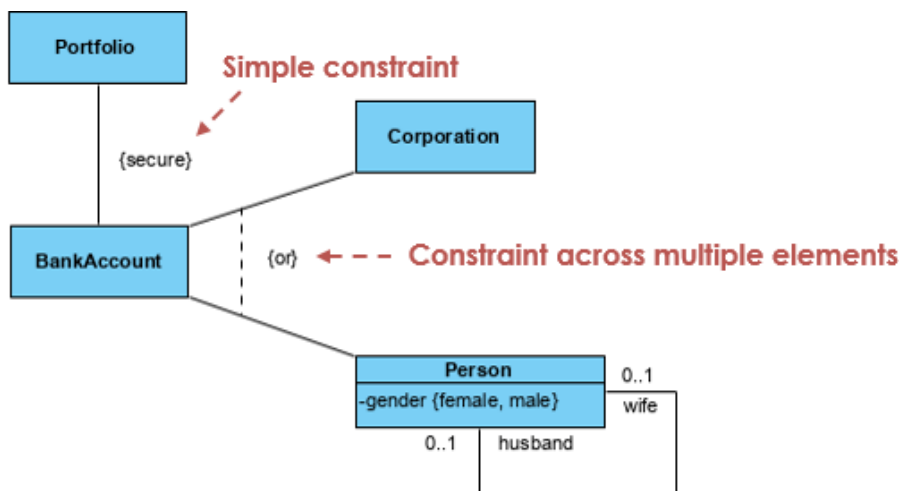
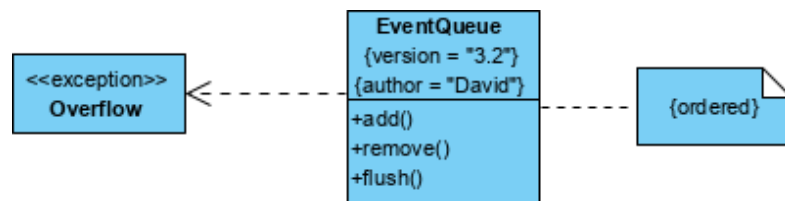
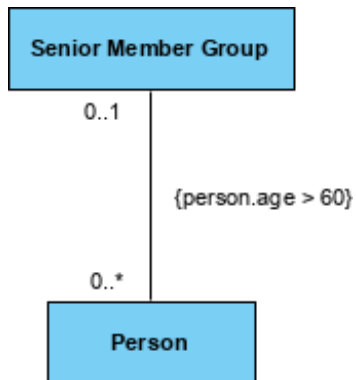
Resumen tipos de asociación en UML

Relación	Función	Notación
Asociación	Describe conexiones semánticas entre objetos individuales de diferentes clases.	
Dependencia o uso	Relaciona clases cuyo comportamiento o implementación afecta a otras clases. No implica que la contenga como atributo, puede crear una instancia o llamar a un método estático (ej: Math.random())	
Generalización	Relaciona a los clasificadores padres con sus hijos. Se usa para la herencia y declaraciones del tipo polimórfico.	
Realización	Relaciona una especificación con con su implementación	
Composición	Las partes: - Sólo eisten asociadas al compuesto - Se crean y se destruyen con él. - Solo se accede a ellas a través del compuesto	
Agregación	Las partes pueden formar parte de distintos agregados. Sin ellas el compuesto sigue teniendo sentido.	

Restricciones en UML

Se ponen entre llaves { }

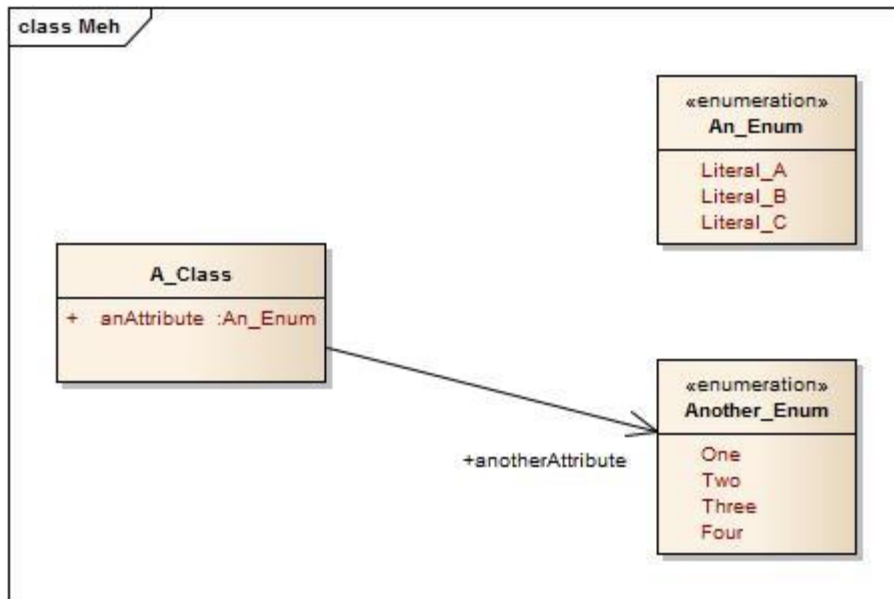
Ejemplos:



Enumeraciones en UML

Son clases normales cuyo nombre tiene precedido en estereotipo <<enumeration>>

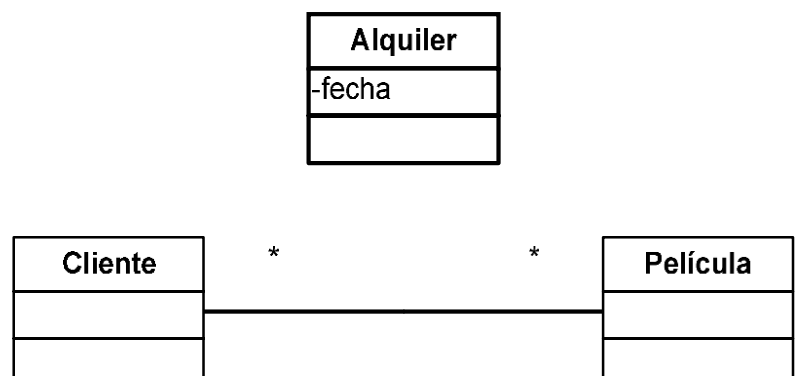
Aquí vemos una clase asociada a 2 tipos de enumeración (una se incluye como atributo y la otra está conectada por una línea)



Clases asociación

Las clases asociación se emplean para indicar que la asociación existente entre dos clases tiene atributos propios. Normalmente, si hay una fecha asociada en la asociación entre una clase y otra, habrá que crear esa nueva clase que surge de la asociación.

Ejemplo: La fecha del alquiler no es un atributo del cliente ni de la película, es algo específico del hecho de alquilar la película.



En realidad, las clases asociación de un diagrama de clases UML son clases convencionales cuyo único papel consiste en relacionar objetos de otras clases (no tienen comportamiento propio). Se modelan como cualquier otra clase, conteniendo un atributo de cada clase que conectan, más los atributos propios (fecha, etc.)

```
class Cliente
...
class Pelicula
...
class Alquiler
{
    private Cliente cliente;
    private Pelicula peli;
    private DateTime fecha;

    public Alquiler (Cliente cliente, Pelicula peli, DateTime fecha)
    {
        this.cliente = cliente;
        this.peli = peli;
        this.fecha = fecha;
    }
}
```

Otro ejemplo:

