Modelado de Clases aplicando UML (Unified Modeling Language): Conceptos Básicos

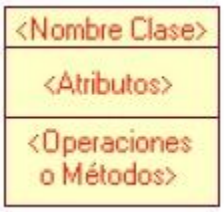
Un diagrama de clases sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de contención.
Un diagrama de clases está compuesto por los siguientes elementos:

- Clase: atributos, métodos y visibilidad.
- Relaciones: Herencia, Composición, Agregación, Asociación y Uso.

Elementos

- **Clase**

Es la unidad básica que encapsula toda la información de un Objeto (un objeto es una instancia de una clase). A través de ella podemos modelar el entorno en estudio (una Casa, un Auto, una Cuenta Corriente, etc.).
En UML, una clase es representada por un rectángulo que posee tres divisiones:



En donde:

- Parte superior: Contiene el nombre de la Clase
- Parte intermedia: Contiene los atributos (o variables de instancia) que caracterizan a la Clase (pueden ser private, protected o public).
- Parte inferior: Contiene los métodos u operaciones, los cuales son la forma como interactúa el objeto con su entorno (dependiendo de la visibilidad: private, protected o public).

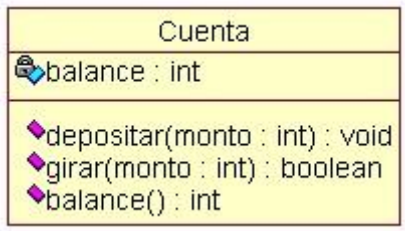
Ejemplo:
Una Cuenta Corriente que posee como característica:

- Balance

Puede realizar las operaciones de:

- Depositar
- Girar
- y Balance

El diseño asociado es:



Atributos y Métodos:

- **Atributos:**

  Los atributos o características de una Clase pueden ser de tres tipos, los que definen el grado de comunicación y visibilidad de ellos con el entorno, estos son:

  - **public** (+, ): Indica que el atributo será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados.
  - **private** (-, ): Indica que el atributo sólo será accesible desde dentro de la clase (sólo sus métodos lo pueden acceder).
  - **protected** (#, ): Indica que el atributo no será accesible desde fuera de la clase, pero si podrá ser accedido por métodos de la clase además de las subclases que se deriven (ver herencia).

- **Métodos:**

  Los métodos u operaciones de una clase son la forma en como ésta interactúa con su entorno, éstos pueden tener las características:

  - **public** (+, ): Indica que el método será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados.
  - **private** (-, ): Indica que el método sólo será accesible desde dentro de la clase (sólo otros métodos de la clase lo pueden acceder).
  - **protected** (#, ): Indica que el método no será accesible desde fuera de la clase, pero si podrá ser accedido por métodos de la clase además de métodos de las subclases que se deriven (ver herencia).

- **Relaciones entre Clases:**

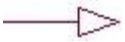
Ahora ya definido el concepto de Clase, es necesario explicar cómo se pueden interrelacionar dos o más clases (cada uno con características y objetivos diferentes).

Antes es necesario explicar el concepto de cardinalidad de relaciones:

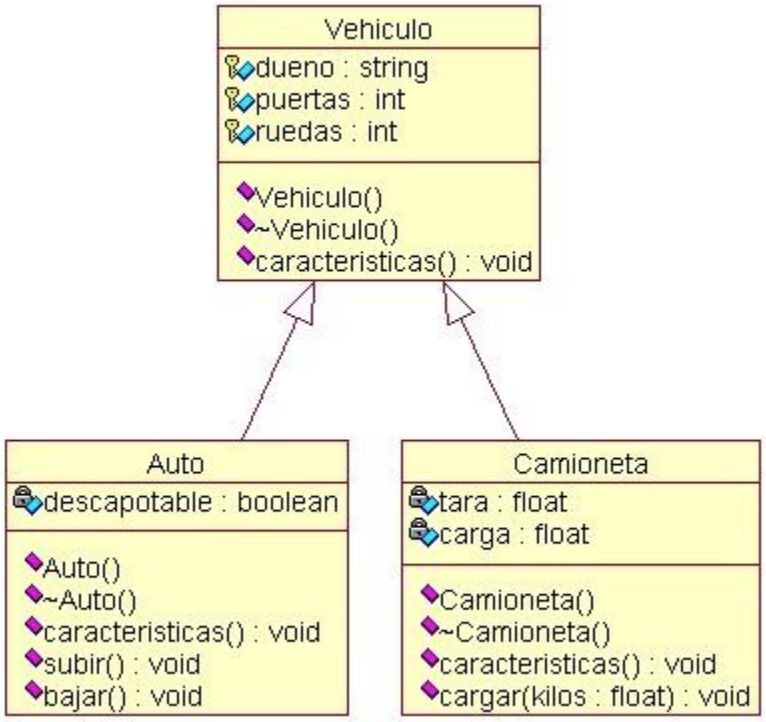
En UML, la cardinalidad de las relaciones indica el grado y nivel de dependencia, se anotan en cada extremo de la relación y éstas pueden ser:

- **uno o muchos**: 1..* (1..n)
- **0 o muchos**: 0..* (0..n)
- **número fijo**: m (m denota el número).

Las relaciones que pueden establecerse entre clases son las siguientes, se indica además su representación en UML:

Relación de Herencia (Especialización/Generalización): 

Indica que una subclase hereda los métodos y atributos especificados por una Super Clase, por ende la Subclase además de poseer sus propios métodos y atributos, poseerá las características y atributos visibles de la Super Clase (public y protected), ejemplo:



En la figura se especifica que Auto y Camión heredan de Vehículo, es decir, Auto posee las Características de Vehículo (Precio, VelMax, etc) además posee algo particular que es Descapotable, en cambio Camión también hereda las características de Vehiculo (Precio, VelMax, etc) pero posee como particularidad propia Acoplado y Carga.

Cabe destacar que fuera de este entorno, lo único "visible" serán los métodos.

En especial el método características, es aplicable a instancias de Vehículo, Auto y Camión, y en cada una de las clases tiene una definición pública específica a la clase a la que pertenece.

Atributos como descapotable no son visibles por ser privados a la clase. Los atributos en POO se indican como privados aplicando la encapsulación de objetos propia del paradigma.

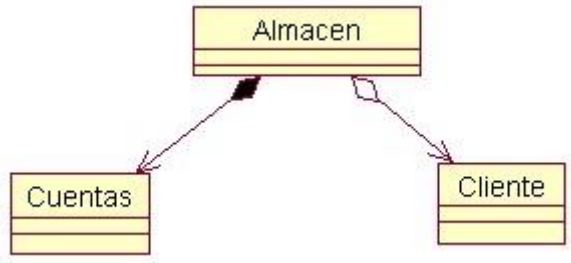
Relación de Agregación: 

Para modelar objetos complejos, no bastan los tipos de datos básicos que proveen los lenguajes: estos tipos son enteros, reales y secuencias de caracteres.

Cuando se requiere componer objetos que son instancias de clases definidas por el desarrollador de la aplicación, tenemos dos posibilidades:

- **Por Valor**: Es un tipo de relación estática, en donde el tiempo de vida del objeto incluido está condicionado por el tiempo de vida del objeto que lo incluye. Este tipo de relación es comúnmente llamada **Composición** (el Objeto base se construye a partir del objeto incluido, es decir, es "parte/todo").
- **Por Referencia**: Es un tipo de relación dinámica, en donde el tiempo de vida del objeto incluido es independiente del que lo incluye. Este tipo de relación es comúnmente llamada **Agregación** (el objeto base utiliza al incluido para su funcionamiento).

Un Ejemplo es el siguiente:



En donde se destaca que:

- Un Almacén posee Clientes y Cuentas (los rombos van en el objeto que posee las referencias).
- Cuando se destruye el Objeto Almacén también son destruidos los objetos Cuenta asociados, en cambio no son afectados los objetos Cliente asociados.
- La composición (por Valor) se destaca por un rombo relleno.
- La agregación (por Referencia) se destaca por un rombo transparente.

  La flecha en este tipo de relación indica la navegabilidad del objeto referenciado. Cuando no existe este tipo de particularidad la flecha se elimina.

Relación de Asociación: 

La relación entre clases conocida como Asociación, permite asociar objetos que colaboran entre sí. Cabe destacar que no es una relación fuerte, es decir, el tiempo de vida de un objeto no depende del otro.

Ejemplo:

Un cliente puede tener asociadas muchas Órdenes de Compra, en cambio una orden de compra solo puede tener asociado un cliente.

Relación de Dependencia o Instanciación (uso): 

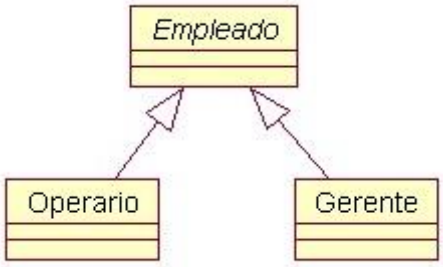
Representa un tipo de relación muy particular, en la que una clase es instanciada (su instanciación es dependiente de otro objeto/clase). Se denota por una flecha punteada.

El uso más particular de este tipo de relación es para denotar la dependencia que tiene una clase de otra, como por ejemplo una aplicación gráfica que instancia una ventana (la creación del Objeto Ventana está condicionado a la instanciación proveniente desde el objeto Aplicación):

Cabe destacar que el objeto creado (en este caso la Ventana gráfica) no se almacena dentro del objeto que lo crea (en este caso la Aplicación).

Casos Particulares:

- **Clase Abstracta**:



Una clase abstracta se denota con el nombre de la clase y de los métodos con letra "itálica". Esto indica que la clase definida no puede ser instanciada pues posee métodos abstractos (aún no han sido definidos, es decir, sin implementación). La única forma de utilizarla es definiendo subclases, que implementan los métodos abstractos definidos.